期末项目展示

NaiveFS

田凯夫、袁方舟

清华大学计算机科学与技术系

2022年6月8日

FUSE

- FUSE(Filesystem In Userspace)是一个面向类Unix计算机操作系统的软件接口

FUSE

- FUSE(Filesystem In Userspace)是一个面向类Unix计算机操作系统的软件接口
- 提高开发效率，简化在内核中实现文件系统的工作量

FUSE

- FUSE(Filesystem In Userspace)是一个面向类Unix计算机操作系统的软件接口
- 提高开发效率，简化在内核中实现文件系统的工作量
- 在用户态实现文件系统会引入额外的特权级切换带来的开销

EXT2

- Linux上的第一款商业级文件系统

EXT2

- Linux上的第一款商业级文件系统
- 规范较为简单，便于实现和扩展

总体架构

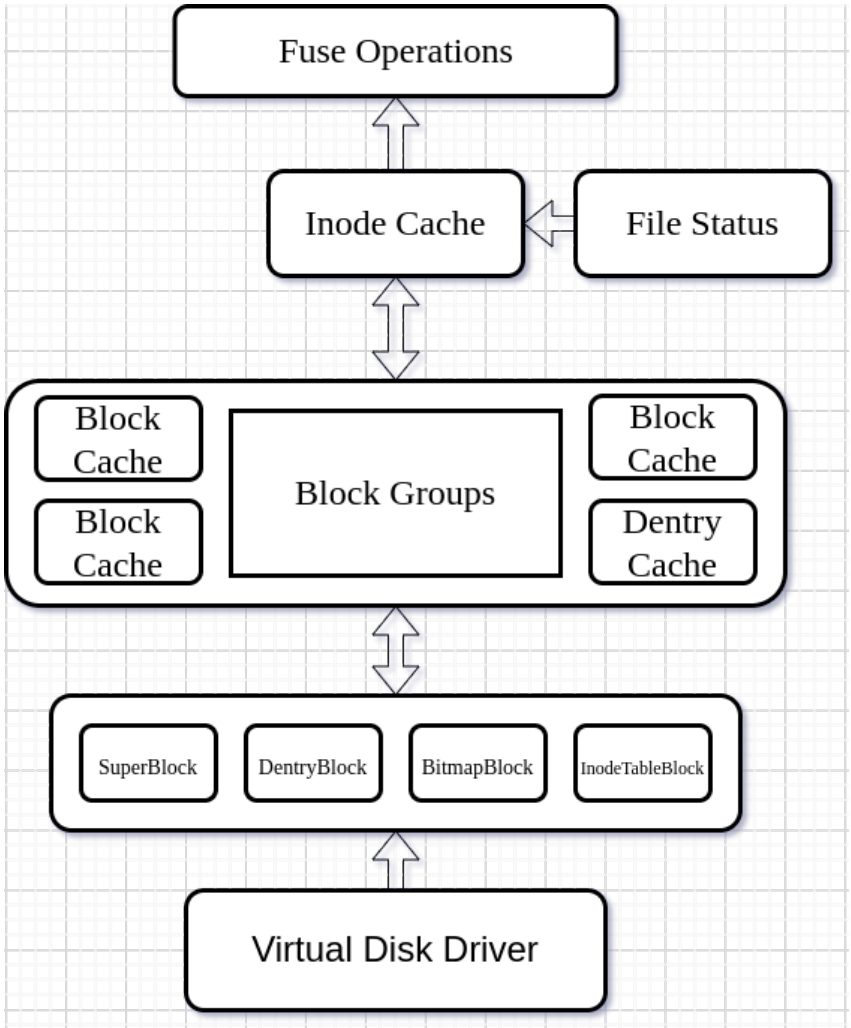
Fuse Operations
Inode Cache
File Status
Block Cache
Block Cache
Block Groups
Block Cache
Dentry Cache
SuperBlock
DentryBlock
BitmapBlock
InodeTableBlock
Virtual Disk Driver

图 1: 总体架构

Block

- 各种Block的基类

Block

- 各种Block的基类
- Block默认大小为**4096B**，在创建时会分配内存空间，在删除时释放分配的内存空间

Block

- 各种Block的基类
- Block默认大小为**4096B**，在创建时会分配内存空间，在删除时释放分配的内存空间
- 处理创建Block时可能的两种情况：1.Block可能是新创建的，虚拟磁盘中的这块区域还未被使用过；2.Block从虚拟磁盘中读取过去写入的数据

Block

- 各种Block的基类
- Block默认大小为**4096B**，在创建时会分配内存空间，在删除时释放分配的内存空间
- 处理创建Block时可能的两种情况：1.Block可能是新创建的，虚拟磁盘中的这块区域还未被使用过；2.Block从虚拟磁盘中读取过去写入的数据
- 提供flush操作，将Block中的数据写入到虚拟磁盘对应的偏移量处

FileSystem

- 创建文件系统时会检查Super Block状态，如果未被初始化，就创建Root Inode作为根目录

FileSystem

- 创建文件系统时会检查Super Block状态，如果未被初始化，就创建Root Inode作为根目录
- 查找：根据提供的路径向下查找，先检查Dentry Cache，再检查Dentry Block

FileSystem

- 创建文件系统时会检查Super Block状态，如果未被初始化，就创建Root Inode作为根目录
- 查找：根据提供的路径向下查找，先检查Dentry Cache，再检查Dentry Block
- 创建：1.查询父目录并获取父目录的Inode；2.遍历Dentry Block；3.分配新的Inode

FileSystem

- 创建文件系统时会检查Super Block状态，如果未被初始化，就创建Root Inode作为根目录
- 查找：根据提供的路径向下查找，先检查Dentry Cache，再检查Dentry Block
- 创建：1.查询父目录并获取父目录的Inode；2.遍历Dentry Block；3.分配新的Inode
- 删除：减少引用计数，若引用计数归零，则删除Inode并清空Cache对应项

优化设计

- **Dentry Cache**：多叉树结构，每一层采用环形链表，父节点指向最新被访问到的节点

优化设计

- **Dentry Cache**：多叉树结构，每一层采用环形链表，父节点指向最新被访问到的节点
- **Block Cache**：使用散列表对Block进行管理，采用LRU替换策略，Block的空间只有在被替换的时候才会释放

优化设计

- **Dentry Cache**：多叉树结构，每一层采用环形链表，父节点指向最新被访问到的节点
- **Block Cache**：使用散列表对Block进行管理，采用LRU替换策略，Block的空间只有在被替换的时候才会释放
- **Inode Cache**：对多个FileStatus共用的inode进行缓存，用读写锁管理并发访问，更新时会同步更新引用它的FileStatus

多线程设计

- 除读写文件和只读操作以外都使用大互斥锁，防止元数据被错误修改

多线程设计

- 除读写文件和只读操作以外都使用大互斥锁，防止元数据被错误修改
- 允许多进程对多个File Handle进行多线程操作

多线程设计

- 除读写文件和只读操作以外都使用大互斥锁，防止元数据被错误修改
- 允许多进程对多个File Handle进行多线程操作
- 在所有的进程之间共享Inode的状态

多线程设计

- 除读写文件和只读操作以外都使用大互斥锁，防止元数据被错误修改
- 允许多进程对多个File Handle进行多线程操作
- 在所有的进程之间共享Inode的状态
- 读写文件时维护**FileStatus**

FileStatus

- 对访问的Inode进行缓存，记录上一次的访问时的位置和Indirect Block等，加速seek

FileStatus

- 对访问的Inode进行缓存，记录上一次的访问时的位置和Indirect Block等，加速seek
- 如果这次正好在上一次的下一个block，则直接读取相应位置的Block ID；否则，从上往下重新读取，如果已经被记录过则不必再读取

FileStatus

- 对访问的Inode进行缓存，记录上一次的访问时的位置和Indirect Block等，加速seek
- 如果这次正好在上一次的下一个block，则直接读取相应位置的Block ID；否则，从上往下重新读取，如果已经被记录过则不必再读取
- Inode缓存由多个进程共享，如果有进程更新了Inode，则其他进程也必须重新读入Inode。

FileStatus

- 对访问的Inode进行缓存，记录上一次的访问时的位置和Indirect Block等，加速seek
- 如果这次正好在上一次的下一个block，则直接读取相应位置的Block ID；否则，从上往下重新读取，如果已经被记录过则不必再读取
- Inode缓存由多个进程共享，如果有进程更新了Inode，则其他进程也必须重新读入Inode。
- 读文件：判断offset是否合法，然后对Inode缓存加共享锁。每次读取新的Block的时候视情况对FileSystem相关部分加锁

FileStatus

- 对访问的Inode进行缓存，记录上一次的访问时的位置和Indirect Block等，加速seek
- 如果这次正好在上一次的下一个block，则直接读取相应位置的Block ID；否则，从上往下重新读取，如果已经被记录过则不必再读取
- Inode缓存由多个进程共享，如果有进程更新了Inode，则其他进程也必须重新读入Inode。
- 读文件：判断offset是否合法，然后对Inode缓存加共享锁。每次读取新的Block的时候视情况对FileSystem相关部分加锁
- 写文件：判断是否需要改动Inode（如写的部分超出当前文件范围），再对Inode缓存加共享锁或互斥锁。

加密

- 使用AES256CBC mode对每个Block和Inode Block加密。

加密

- 使用AES256CBC mode对每个Block和Inode Block加密。
- 在Super Block里设置一个特定的字符串，进入文件系统时对这个字符串解密

加密

- 使用AES256CBC mode对每个Block和Inode Block加密。
- 在Super Block里设置一个特定的字符串，进入文件系统时对这个字符串解密
- 如果解密出来是期望的结果就说明密码正确，否则密码错误。

权限检查

- 使用EXT2的对应字段，按照权限规则检查

DEMO

- 搭建运行环境
- 实现底层功能
- 实现Fuse相关函数
- 测试、优化性能
- 拓展实现

Thanks!