基于数据库的文件系统设计与实现

陈林峰　导师: 陆慧梅

北京理工大学

2023 年 5 月 23 日

目录

论文检测结果

- 最终版查重结果:2.8%
- 盲审结果
  - A+(92)
  - A(87)
- 形式审查通过，参加评优

文件系统与数据库

- 数据库与文件系统承担同样的功能，但数据库更注重数据的一致性、可靠性、安全性
- 文件系统中许多新的思路和想法与数据库中已有的不谋而合
- 在文件系统中添加数据库中的功能需要大量的重复工作

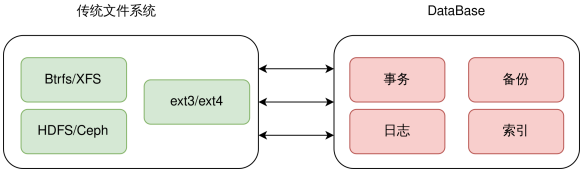


图 1: 数据库与文件系统

相关的工作

- File System Interfaces to Databases
  - IFS (基于关系型数据库 POSTGRES 实现的文件系统)
- File Systems Built Using a Database
  - Oracle iFS (Oracle)
  - AMINO (Berkeley Database)
- File Systems with Database Features (事务)
  - exF2FS
  - DurableFs

目的

- 重用数据库的基础设施
- 利用数据库本身具有的高效索引、事务特性，增强文件系统的搜索效率、安全性
- 简化文件系统实现，提高文件系统的可扩展性

毕设项目组成

- Alien: 使用 rust 实现的简单类 linux 操作系统，验证将数据库文件系统移植到操作系统内核的可能性[1]
- jammdb: key-value 数据库，本文选择使用的数据库[2]
- rvfs: rust 写的 vfs 框架，主要参考 linux 中的 vfs, 用在 Alien OS 中[3]
- dbfs: 本文设计实现的数据库文件系统，同时包含了用户态 fuse 实现 [4]
- dbop: 数据库操作抽象，一种在操作系统中导出数据库接口的方法

[1] Alien OS:https://github.com/Godones/Alien
[2] jammdb:https://github.com/Godones/jammdb
[3] rvfs:https://github.com/Godones/rvfs
[4] dbfs:https://github.com/Godones/dbfs2

项目结构图

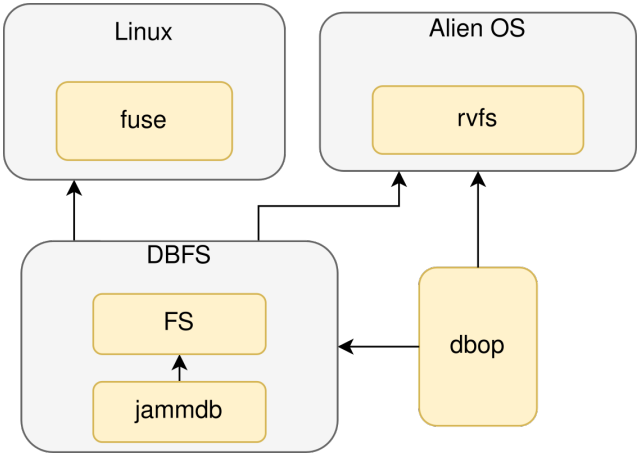


图 2: 项目关系图

- 嵌入式、单文件
- ACID
- 并发读取与单个写入
- B+Tree、mmap
- key-value and key-bucket

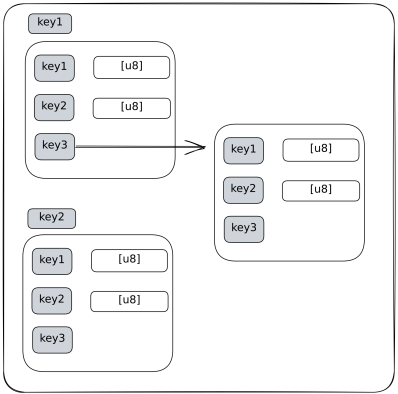


图 3: jammdb

元数据管理与目录树构建-Everything is key-value

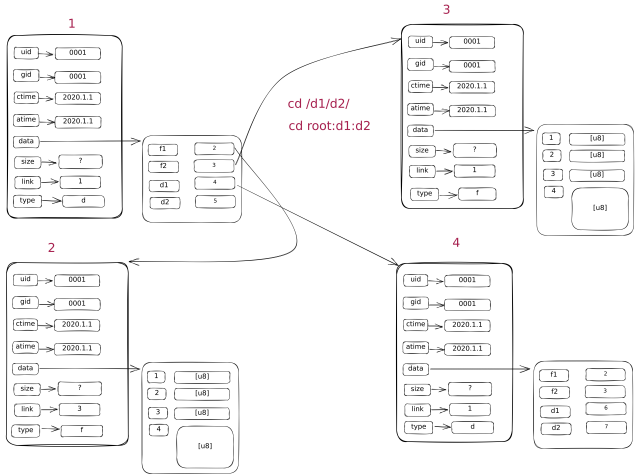


图 4: 目录树与元数据

文件数据存储-Everything is key-value

表 1: 目录存储方式

key	value	含义
data: name1	111	name1 文件在全局空间的 bucket 标识为 111
data: name2	222	name2 文件在全局空间的 bucket 标识为 222

表 2: 软链接数据存储方式

key	value	含义
data	path	软链接路径信息

表 3: 文件数据存储方式

key	value	含义
data: 0	[u8;blk_size]	第一块数据
data: 1	[u8;blk_size]	第二块数据
data: 2	[u8;blk_size]	第三块数据

数据库的接口导出

```
let tx = db.tx()
let bucket = tx.get_bucket()
tx.delete_bucket()
tx.get_bucket()
tx.create_bucket()
bucket.delete_bucket()
bucket.get_bucket()
bucket.get_kv()
bucket.put()
bucket.delete()
bucket.create_bucket()
```

```
pub struct AddKeyOperate {
    pub map: BTreeMap<String, Vec<u8>>,
}
pub struct DeleteKeyOperate {
    pub keys: Vec<String>,
}
pub struct AddBucketOperate {
    pub key:String,
    pub other:Option<Box<OperateSet>>
}
pub struct StepIntoOperate {
    pub key:String,
    pub other:Option<Box<OperateSet>>
}
```

图 5: 数据库接口导出

核心数据结构

- 核心是数据库实体 DB，需要外部使用者进行初始化
- DbfsAttr 、DbfsStat、DbfsError、DbfsPerm 这类结构，其作用主要是完成通用接口与上层适配接口的转换
- Cache pool、Globalinfo 这类接口，其主要作用是记录某些文件操作信息，以及避免一些不必要的内存分配开销

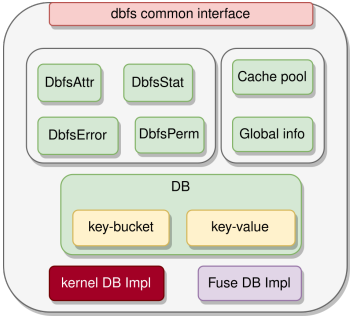


图 6: 核心数据结构

接口设计

- 存储介质: 普通文件是块设备
- 数据库层是存储算法的载体，其负责组织数据的存储，管理文件系统的所有信息
- DBFS 层构建文件系统，提供了通用的接口
- 最上层是 DBFS 最终的表现形式，可以适配 fuse 接口或者接入内核的 VFS 接口

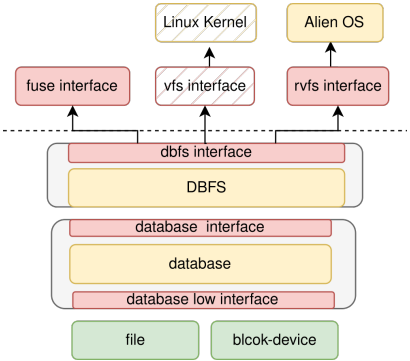


图 7: 接口设计

Fuse 实现

- 用户态 DBFS 可以使用标准性能测试工具
- 检验文件系统实现正确性
- 使用普通文件作为底层存储设备
- 适配 fuse 接口

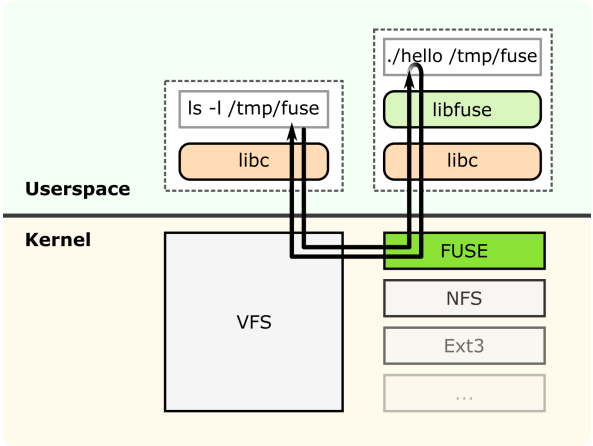


图 8: fuse

Alien OS 实现

- 物理内存、虚拟内存管理
- 中断
- 进程管理
- 文件系统
- 虚拟文件系统
- 外设

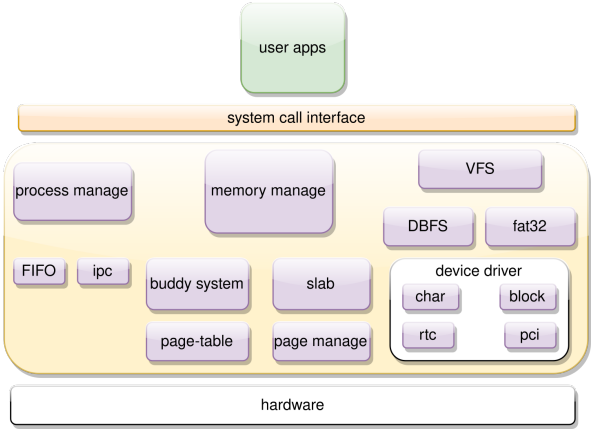


图 9: Alien OS

- 屏蔽文件系统之间的差异
- 提供统一的系统调用接口
- 使用 rust 实现的 vfs 框架
- 参考 linux 的 vfs
- 内置 ramfs 支持
- 添加 fat32 的支持

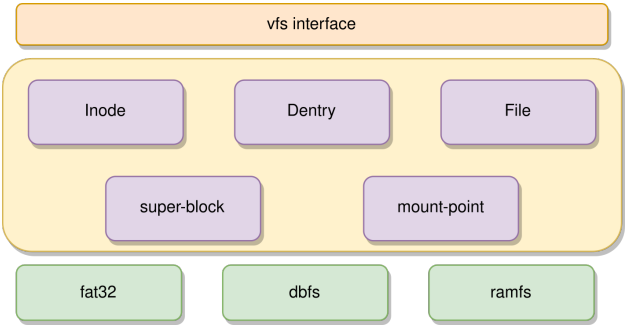


图 10: VFS 实现

DBFS 内核移植

- 虚拟文件 FakeFile
- 缺页处理动态加载
- 块设备映射到一段连续的虚拟地址空间
- mmap 与 file 接口共用同一抽象

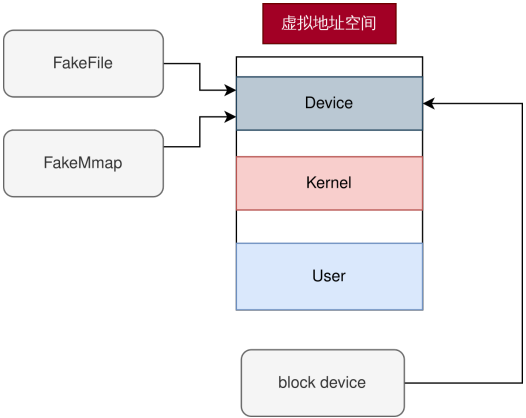


图 11: 内核移植

pjdfstest 文件系统正确性测试

- 检验系统的正确性
- POSIX 兼容性测试通过率 92%
- 少数错误处理出错
- ioctl 支持不完善

test-set	interface	pass/all	error/all	error
chflags	chflags(FreeBSD)	14/14	0	linux 下不工作
chmod	chmod/stat/symlink/chown	321/327	26/327	chmod 实现有误
chown	chown/chmod/stat	1280/1540	260/1540	chmod 实现有误
ftruncate	truncate/stat	88/89	1/89	Access 判断出错
granular	未知	7/7	0	linux 下不工作
link	link/unlink/mknod	359/359	0	无
mkdir	mkdir	118/118	0	无
mkfifo	mknod/link/unlink	120/120	0	无
mknod	mknod	186/186	0	无
open	open/chmod/unlink/symlink	328/328	0	无
posix_fallocate	fallocate	21/22	1/22	Access 判断错误
rename	rename/mkdir/	4458/4857	399/4857	错误返回值处理与逻辑错误
rmdir	rmdir	139/145	6/145	错误处理，权限检查
symlink	symlink	95/95	0	无
truncate	truncate	84/84	0	无
unlink	unlink/link	403/440	37/440	mknod 中的 socket, 错误处理
utimensat	utimens	121/122	1/122	权限检查
		7943/8674	731/8674	92%

图 12: pjdfstest 测试

mdtest 元数据性能测试

- 部分测试达到了与 ext3 相同的水平
- rename 操作远远高于另外两个
- 充分利用数据库的数据结构优势加速查找过程
- 在 fuse 的实现中增加缓存结构

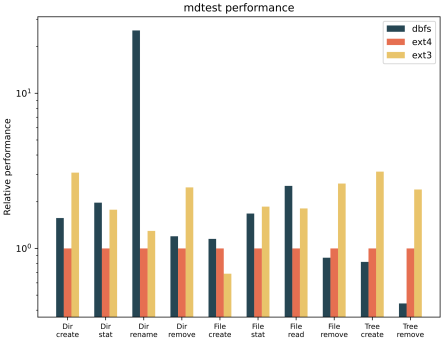
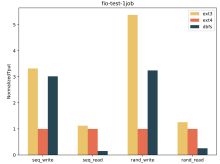
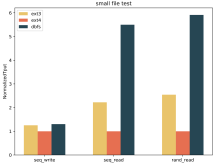


图 13: mdtest

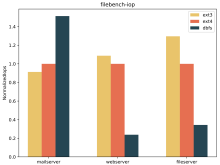
Fio 读写测试与 Filebench 实际应用负载测试



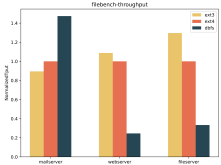
(a) fio 大文件读写测试-NoCache



(b) fio 小文件读写测试-WithCache



(c) filebench-iop



(d) filebench-throughput

Alien OS 中 FAT32 与 DBFS 的读写测试

表 4: Alien OS 测试结果

(MB/s)/FS	DBFS	FAT32
顺序写 (1MB)	32MB/s	15MB/s
顺序写 (Re)	384MB/s	104MB/s
顺序读 (1MB)	1000MB/s	110MB/s
顺序读 (4KB)	27MB/s	50MB/s
随机写 (1MB)	386MB/s	56MB/s
随机读 (1MB)	1066MB/s	58MB/s

总结和展望

- 设计并实现一个基于 key-value 类型的数据库文件系统 DBFS
- 将 DBFS 支持 linux 的 fuse
- 将 DBFS 移植到 Alien OS 中
- 完成用户态和内核态两种情况下的测试与分析

- 分析 DBFS 潜在的性能瓶颈
- 基于更强大的数据库实现 DBFS
- 移植 DBFS 到 linux 内核当中

Thank you !