中期报告

本报告总结了在毕设项目中已经完成的工作,并梳理了当前遇到的问题，以及这些问题的初步解决方案。

毕业设计（论文）主要研究内容、进展情况及取得成果

研究内容

文件系统和数据库是两个不同的概念，但它们有密切的关系。文件系统是操作系统中用于管理计算机中存储数据的方式。它负责文件的存储、检索和管理，并提供文件的访问和保护机制。文件系统通常使用树形结构来组织文件和目录，使得用户可以方便地浏览和访问存储在计算机中的文件。而数据库是一种专门用于管理和存储大量结构化数据的软件系统，它也可以帮助用户存储、管理和访问数据，并提供高效的数据查询和处理功能。数据库通常使用表格形式来组织数据，这样可以使得数据的存储和查询更加方便和高效。尽管文件系统和数据库是两个不同的概念，但是它们之间也存在一些相互关联的概念，数据库系统通常会使用文件系统来存储它们的数据文件，而文件系统的访问控制机制也可以用于保护数据库中的数据。此外，许多数据库系统还提供了文件系统功能，允许用户在数据库中存储和检索文件和其他二进制数据。

文件系统和数据库是在相互借鉴中发展的，但是一个有趣的现象是绝大多数的数据库系统都是基于文件系统实现，那既然两者的功能如此类似，为什么不能反过来让文件系统基于数据库实现呢。这样一来，在文件系实现中就不需要重新引入来自数据库的特性了，因为基于数据库的文件系统已经自然地拥有了这些特性。

基于数据库的文件系统设计与实现，是一个基于数据库技术的文件系统实现方案。其可以利用数据库的高效管理和查询能力来存储和管理文件数据，能够更好地支持大规模、多类型数据的管理和查询。此外，基于数据库的文件系统还能够支持多用户、多任务的并发操作，提高系统的可靠性和性能。

本研究旨在通过将文件系统和数据库系统相结合，充分发挥两者的优势，实现高效、可扩展、易管理的文件存储与管理系统。具体地，通过将文件和相关的元数据存储在数据库中，利用数据库系统的索引机制实现高效的文件检索和访问，同时设计文件在数据库中的存储方式，实现文件的高效存储和管理，在用户态，我们也提供数据库操作的直接抽象使得用户也可以直接使用数据库的接口进行数据的存储和访问，这种设计不仅可以提高系统的性能和可扩展性，还可以提高系统的可靠性和安全性。

基于数据库的文件系统设计与实现的研究具有重要的理论和实际意义。它可以为用户数据管理和存储提供一种新的解决方案，同时还能扩展文件系统的功能。

基于数据库的文件系统设计与实现的主要研究内容是将数据库的存储和管理思想结合到文件系统中，设计并实现一个具有数据库特性的文件系统，实现对文件的高效管理和查询。具体研究内容包括：

1. 数据库技术与文件系统结合：将数据库的基本概念、数据结构、数据存储和管理技术与文件系统的数据组织、存储和访问方式相结合，构建一个具有数据库特性的文件系统。
2. 文件管理与查询优化：设计一套文件管理系统，包括文件的存储、读取、修改、删除等操作，并优化文件的查询性能，通过对文件元数据的索引和优化算法等手段，实现对文件的快速查询和检索。
3. 数据一致性与事务管理：研究文件系统和数据库之间的数据一致性问题，实现事务管理功能，确保数据的完整性和一致性。
4. 安全性与权限管理：设计一套安全性和权限管理系统，包括文件访问控制、用户身份验证和授权等功能，保护文件系统的安全性和隐私。
5. 系统实现与性能测试：基于上述研究内容，设计并实现一个完整的基于数据库的文件系统原型，并进行性能测试和优化，验证其可行性和实用性。

本项目的所有实现使用`rust`语言完成，在实现中，本文没有选择功能强大的数据库来作为文件系统实现的底层支持，而是选择了 一个简单的`key-value`数据库。同时，因为使用`rust`来完成所有实现，而将这些实现直接放到`linux`中和还不现实，因此本文为此实现了一个简单的操作系统。同时为了获得它的性能数据，我们选择了使用`rust`实现的`fat32`文件系统进行对比。因此，本项目总共会包含以下几个项目:

1. Alien: 使用`rust`实现的简单操作系统
2. jammdb: key-value数据库
3. fat32-vfs: fat32文件系统
4. rvfs: rust写的vfs框架
5. dbfs: 数据库文件系统
6. dbop: 数据库操作抽象

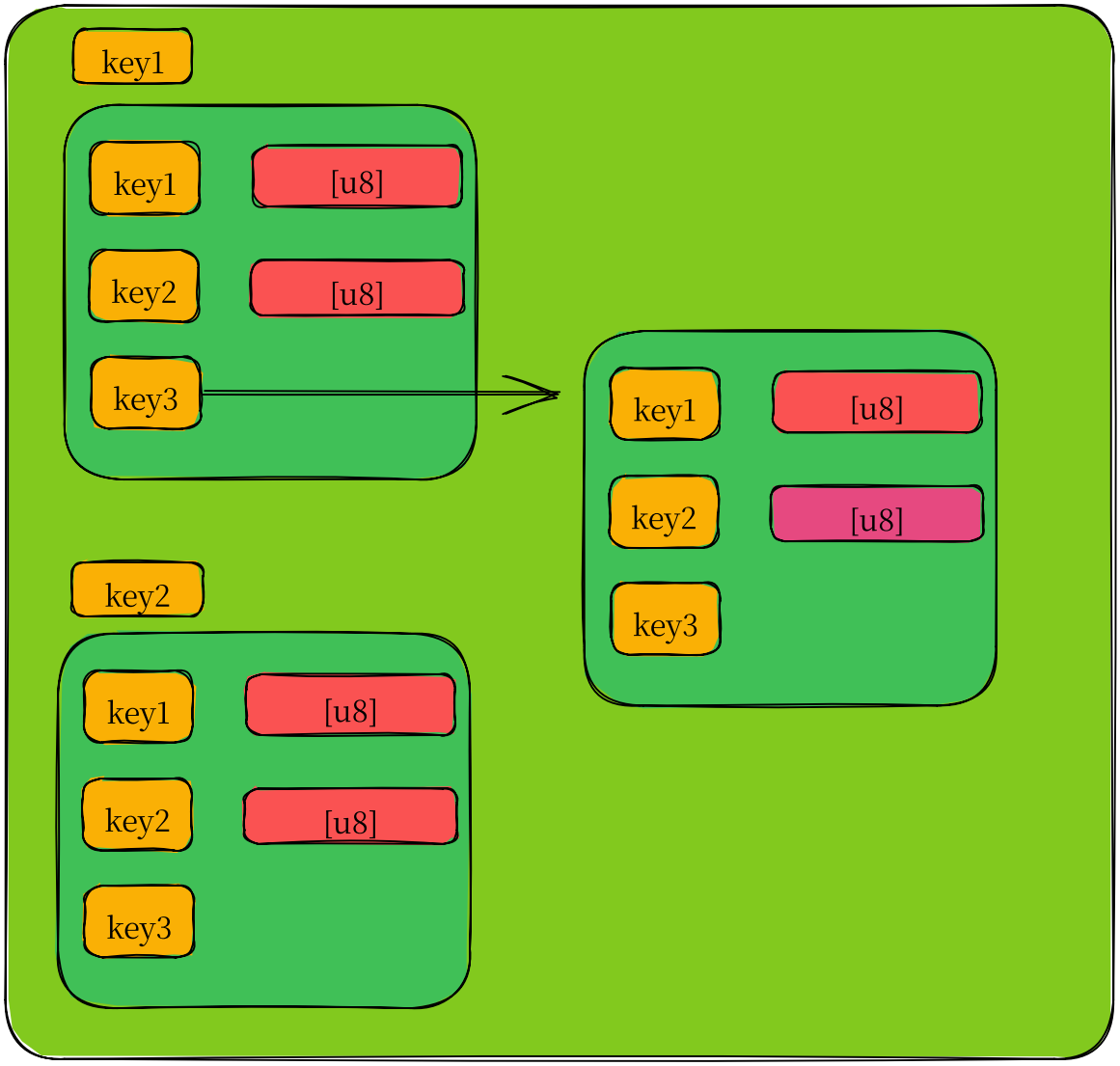
Alien

这是一个使用`rust`实现的简单操作系统，其基于qemu模拟器，支持`riscv64`位平台。这个简单的操作系统原本是用来探索模块化os的，在比设选择了实现文件系统后本文选择将其作为文件系统以及应用程序的运行平台。目前已经完成的功能包括内存管理，虚存管理，外部设备，中断，进程管理，文件系统，虚拟文件系统等，已经支持数十个系统调用，同时支持rust编写的应用程序和c语言程序。

Jammdb

jammdb是一个嵌入式、单文件的key-value数据库，其提供ACID特性，支持多个并发读取和多个单个写入。所有的数据被组织成一棵B+树，随机和顺序读取速度很快。其对文件的操作基于内存映射。选择这个数据库作为dbfs实现的原因是简单，因为复杂的数据库难以不仅难以移植到裸机平台，要将其放到内核态需要大量的工作，但简单带来的一个坏处就是数据库的功能不够强大，并且数据库本身没有对磁盘设备作出相应的优化，对于毕设来说，我们主要进行dbfs的探索和尝试，因此不必选择复杂的数据库。为了在no_std环境下使用jammdb，本文fork了原项目仓库，删除了原项目中依赖的标准库或操作系统接口，例如一些标准库算法以及mmap系统调用接口，并将这些依赖重新定义。这样一来用户只需要提供这些接口的实现，就可以使用此数据库，而为了检查作出的修改是否影响了原实现的正确性，我们在其内部实现了一个内存模拟的文件，并实现抽象出来的接口，在通过了所有测试后，可以说明我们的修改没有引入错误。

数据库的数据结构如下所示:



数据库的内部基于桶`bucket`实现。如上面所示，该数据库的基本结构由一个个处于全局空间的`bucket`组成，`bucket`可以存储普通的key-value数据，这里key和value都是[u8]数组，同时也可以存储嵌套的`bucket`数据结构。一个`bucket`是由一棵B+树构成，b+树将大小为4k或者其它大小的页面组织起来存储数据，这里的页面与传统的磁盘块类似，只是因为数据库使用内存映射而将存储结构设置为页大小。

fat32-vfs

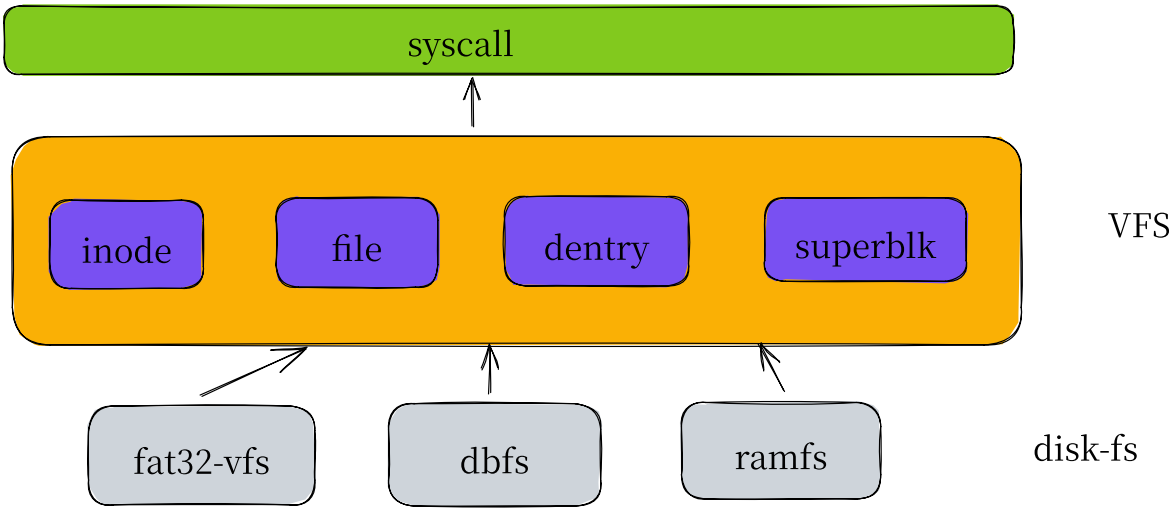
这是一个使用`rust`实现的基于vfs接口的fat文件系统，目前使用rust实现的文件系统实例并不多，而我们实现的dbfs需要与其他文件系统进行对比，虽然笔者已经实现了一个fat32文件系统，但为了更有说服力，还是选择了这个使用人数比较多的实现。本文对原项目进行一些细节修正，然后实现了接下来提到的vfs接口，这样一来，这个fat32就可以接入到操作系统中了。

在实现中，因为fat系列的文件系统并没有inode的抽象，为了与vfs接口层进行融合，本文使用了vfs层的Inode结构的data字段保存了一些索引信息，从而加快文件的查找速度。

fat32实现了vfs的一部分接口，因为fat32本身不支持很多功能，比如链接文件，属性扩展等，因此其实现要比dbfs简单。

rvfs

rvfs是一个类似于linux的vfs的框架，其为不同的底层文件系统提供统一的抽象。

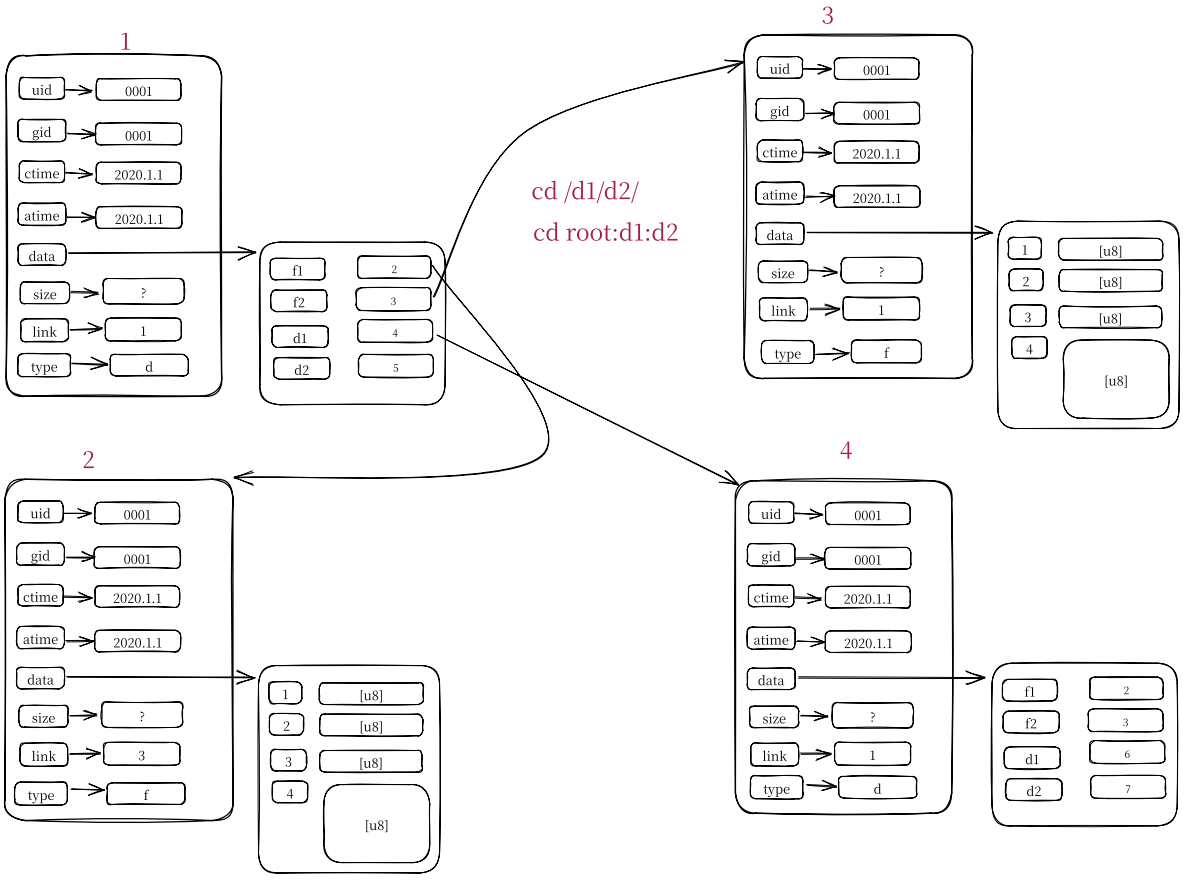


设计上，其主要包含4个数据结构,`Inode`、`File`、`Dentry`、`Superblock`以及这些数据结构相关的操作。实现此项目的原因是我们需要将文件系统在上文提到的操作系统中使用，如果没有vfs的实现，那对fat32和dbfs的操作就需要分别进行处理，并且缺乏这个类似缓存层的实现会导致用户存取和访问文件速度变慢，降低文件系统实现的性能。

vfs的实现包含了一个简单的内存文件系统，可以作为初始挂载点，将其他文件系统挂载在上面。目前vfs实现了linux中的大部分常用接口，包括诸如读写、链接、属性扩展、文件状态、重命名、截断等。

dbfs

dbfs是本文的工作目标，但其工作量却比其他工作少，为了完成数据库文件系统的实现，本文对jammdb的实现进行了分析，在确定了数据库的基本数据结构、以及如何组织数据后，本文根据这个数据结构提供了一种基于全局空间`bucket`的文件系统设计:



对于一个路径`/d1/dd1/f1`来说，VFS会依次解析`/`、`d1`、`dd1`、`f1`，即从根目录下查找得到目录`d1`，再从`d1`中查找得到目录`dd1`,如果dbfs需要在vfs的框架下运行，就需要定义良好的数据结构从而可以完成查找文件/目录的功能。这种设计是为了满足linux中VFS的路径解析算法。

在这种设计下，可以很快速地完成文件系统的实现，因为有了数据库作为基础，文件系统的存储和索引都是基于键值对。但为了更好的性能，我们也对数据的存储作了一定的重新设计，以避免数据库频繁的键值对删除添加操作。

更近一步地，我们探讨了数据库的两个不同视角，一个视角就是使用数据库的数据结构组织成文件系统，另一个视角就是用户直接使用数据库存储文件，在这种视角下，可以一定程度上去除传统的文件/目录的层级组织结构。

详细的设计可以查看文末给出的连接。

进展情况

目前，上述提到的几个项目基本已经完成，在Alien操作系统中，也已经引入了vfs框架，接入了fat32-vfs和dbfs的实现。

为了检验这些文件系统和vfs的正确性，本文在Alien中添加了许多文件系统相关的系统调用，并在用户态编写了应用程序进行检验，实验证明，本文的实现没有难以解决的错误。

在用户态程序中，本文编写了顺序读写性能与随机读写性能的测试。

顺序读性能

```
// 测试条件
const FILE_SIZE: usize = 1024 * 1024 * 10; //10MB
const BLOCK_SIZE: usize = 4096;
```

```
// 测试结果
// fat32
time cost = 12777ms, read speed = 820KB/s
// dbfs
time cost = 22812ms, read speed = 459KB/s
```

顺序写性能

```
// 测试条件
const FILE_SIZE: usize = 1024 * 1024 * 10; //10MB
const BLOCK_SIZE: usize = 1024;
```

```
// 测试结果
time cost = 12632ms, write speed = 830KB/s
time cost = 25085ms, write speed = 418KB/s
```

随机读写性能

1. 打开文件：使用文件 I/O 函数打开测试文件，并将其指针保存在变量中。
2. 生成随机数：使用随机数生成函数生成随机数，用于确定读取或写入文件的位置和大小。
3. 进行随机读写：使用文件 I/O 函数对文件进行随机读写，根据生成的随机数确定读取或写入的位置和大小。要进行随机读写测试，应使用随机位置和随机大小进行读写操作，而不是顺序读写操作。
4. 记录测试结果：在测试期间，应记录每个操作的完成时间、读取或写入的位置和大小等信息，以便后续分析测试结果。

5. 分析测试结果：测试完成后，可以分析测试结果以了解文件系统的随机读写性能。可以计算测试期间每秒钟完成的操作数、延迟、带宽等指标，并将它们与其他文件系统进行比较。

```
// 测试条件
const FILE_SIZE: usize = 1024 * 1024 * 16;
// 16MB
const BLOCK_SIZE: usize = 4096;
// 随机查找次数
const ITER: usize = 1000_0;
```

```
// fat32测试结果
Random read test:
Elapsed time: 155409ms
Read: 39997KB
Throughput: 257.367002152385KB/s
Operations: 64.3463377281882ops/s

Random write test:
Elapsed time: 153839ms
Write: 40000KB
Throughput: 260.01209056221114KB/s
Operations: 65.00302264055279ops/s
```

```
// dbfs测试结果
Random read test:
Elapsed time: 111441ms
Read: 39992KB
Throughput: 358.8639392144722KB/s
Operations: 89.73358099801689ops/s

Random write test:
Elapsed time: 132381ms
Write: 40000KB
Throughput: 302.15816469130766KB/s
Operations: 75.53954117282692ops/s
```

通过测试结果可以看到，在顺序读写性能上，dbfs较fat32的性能还有较大的差距，在随机读写上dbfs则会表现更好一点。

存在的问题和拟解决方案

目前项目主要存在的问题是如何测试文件系统的性能?

对于这个问题有两种方案可以选择:

1. 使用标准的测试集进行测试，这些测试集主要来自linux文件系统项目或者业界公认的测试，但这个方案的困难在于这些测试对操作系统的要求比较高，可能难以在我们实现的系统上运行。

2. 按照一般的测试方法，自己编写测试程序。文件系统的测试一般会测试顺序读写性能，随机读写性能，压力负载等，可以按照这个思路编写测试。这种方案的优点在于我们可以根据当前操作系统提供的功能进行实现，而不需要实现一些复杂的系统调用，但缺点就是说服力不强，因为这些测试只有我们使用。

在第一种方案中，想要使用linux上的标准测试工具对文件系统功能以及性能作测试，一个可行方案是将实现的dbfs接入fuse文件系统的接口，因为想要直接将实现的dbfs放到内核当中还存在很多困难。

Fuse（Filesystem in Userspace）是一个用户空间文件系统接口，它允许非特权用户在不修改操作系统内核的情况下创建自己的文件系统。Fuse提供了一种通用的机制，用于将用户空间的代码与内核文件系统接口连接起来，从而实现自定义文件系统的开发。

Fuse文件系统工作流程如下：

1. 应用程序调用标准库中的系统调用，例如open()，read()等。
2. 应用程序的请求传递到Fuse内核模块，该模块负责将请求路由到特定的Fuse文件系统实现。
3. Fuse文件系统实现处理请求并执行相关的操作，例如读取或写入文件。
4. 文件系统操作的结果返回给Fuse内核模块。
5. 内核模块将结果返回给应用程序，仿佛这些操作是直接由内核处理的一样。

Fuse的优点在于，它允许用户空间程序使用一种通用的API来实现自定义文件系统。这使得开发人员可以更轻松地开发和测试文件系统，而无需担心对内核进行修改。此外，Fuse还提供了一些其他功能，例如访问远程文件系统，加密文件系统和网络文件系统等。

Fuse文件系统的缺点在于，由于所有文件系统操作都是通过用户空间代码执行的，因此对于某些高负载应用程序，它可能会导致性能问题。

在将dbfs实现fuse定义的接口后，标准的linux文件性能测试工具就可以对dbfs进行测试，同时还可以与linux中一些主要的文件系统进行对比，比如ext4的fuse实现。

第二种方案，因为是自己编写的测试程序，只能在本文中的Alien运行且只能测试dbfs与fat32之间的性能，并且由于是在qemu环境下，与实际的性能数据存在一定的差异。

综上所述，本文拟采用的解决方案是在自己编写测试程序进行测试的同时为dbfs实现fuse的接口，并在编写的os上获得性能数据。

下一步研究任务与进度安排

接下来主要完成的工作包含几个个部分

- [ ] 编写测试程序并在`qemu`模拟器上完成测试
    - [ ] 顺序读写性能
    - [ ] 随机读写性能
    - [ ] 模拟收发邮件进行大批量小文件测试
- [ ] fuse接口实现
    - [ ] 重构dbfs的接口，使其既可以支持Alien操作系统的vfs接口，同时也能方便fuse接口的实现
    - [ ] 使用标准工具验证功能和性能
- [ ] 完成论文初稿

进度安排:

- 在4.25号前完成测试程序的编写并在Alien中进行测试获取数据
- 在4.30号前完成fuse接口的基本实现并能完成一些基本测试
- 在4.30号前完成论文的初稿撰写
- 在5.5号前完成所有fuse的接口实现，可以在linux中进行测试