



revoFS

proj209-Linux-Custom-Filesystem

一、目标描述

revoFS 意为 revolution + FileSystem，目标是设计并实现一个Linux文件系统，该文件系统能够进行文件和目录的读写操作。我们将创建一个Linux内核模块，该模块将新创建的文件系统的操作接口与VFS进行对接，并实现新的文件系统的superblock、dentry、inode的读写操作。此外，我们还将设计并实现一个用户态应用程序，该程序能够将一个块设备（可以用文件模拟）格式化成我们设计的文件系统的格式。

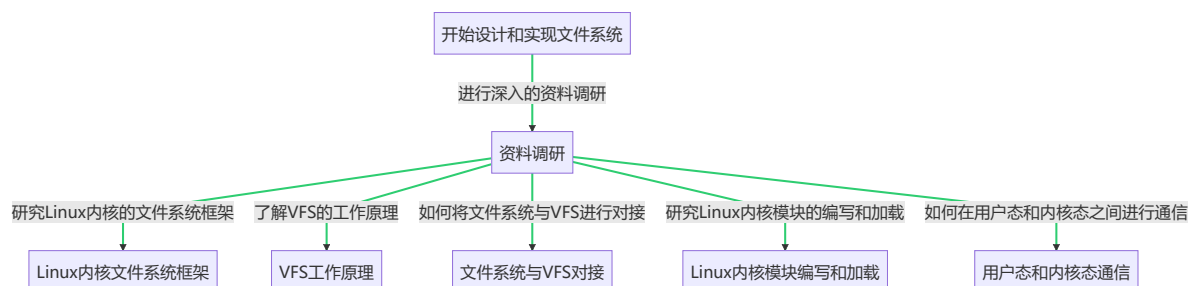
二、比赛题目分析和相关资料调研

在开始设计和实现我们的文件系统之前，我们进行了深入的资料调研。我们研究了Linux内核的文件系统框架，了解了VFS（Virtual File System，虚拟文件系统）的工作原理，以及如何将我们的文件系统与VFS进行对接。VFS是Linux内核中的一个重要组件，它提供了一个抽象层，使得用户程序可以透明地访问各种不同类型的文件系统。VFS支持多个文件系统。Linux内核完成大部分工作，而文件系统特定的任务则委托给各个文件系统通过处理程序来完成。内核并不直接调用函数，而是使用各种操作表，这些操作表是每个操作的处理程序的集合（实际上是每个处理程序/回调的函数指针的结构）。

我们还研究了Linux内核模块的编写和加载，以及如何在用户态和内核态之间进行通信。Linux内核模块是一种可以动态加载和卸载的内核代码，它可以在不重启系统的情况下添加或删除内核功能。用户态和内核态的通信是操作系统设计中的一个重要问题，我们通过研究系统调用、文件系统接口等技术，了解了如何在用户态程序和内核态模块之间传递信息。

在调研过程中，我们参考了许多开源的Linux文件系统项目，例如 ext4、XFS、Btrfs 等。这些项目的源代码为我们提供了宝贵的参考资料，帮助我们理解了如何设计和实现一个功能完备的文件系统。

此外，我们还阅读了大量的技术文档和论文，包括Linux内核文档、文件系统相关的RFC文档、以及关于文件系统设计和实现的学术论文。这些资料为我们提供了深入的理论知识和实践经验。



三、系统框架

revoFS项目的系统框架主要由两部分组成：**内核态的文件系统模块**和**用户态的应用程序**。

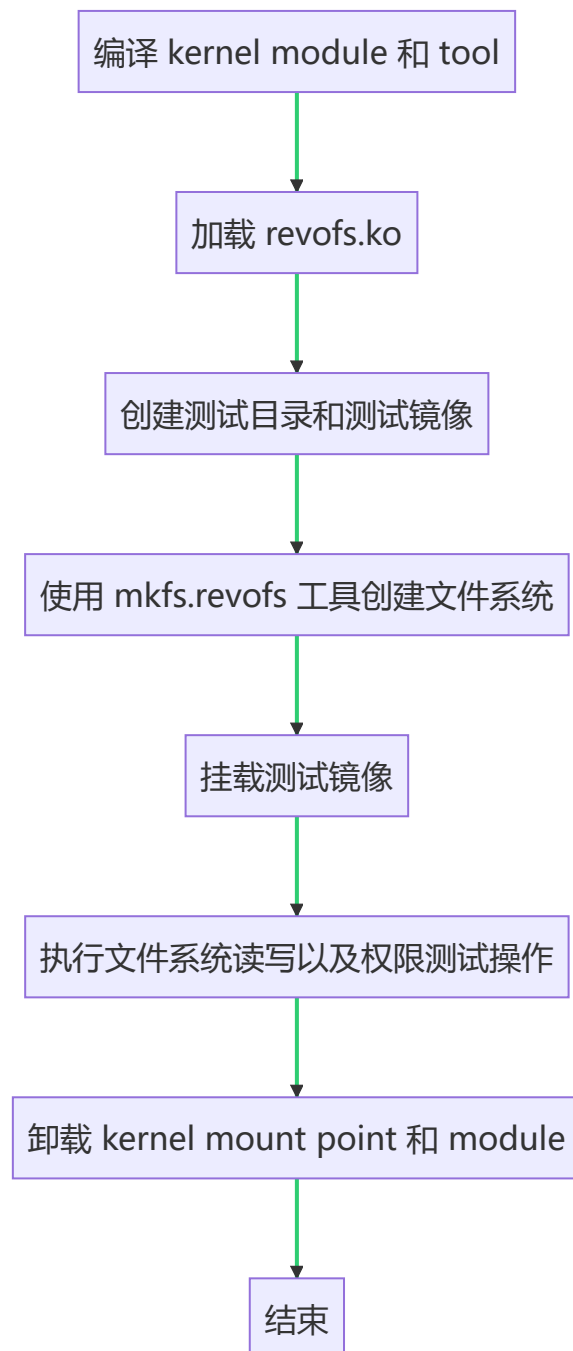
1. 内核态的文件系统模块

文件系统模块是运行在Linux内核态的部分，它负责处理文件和目录的读写操作。这个模块是作为Linux内核模块实现的，可以动态地加载和卸载。文件系统模块实现了新的文件系统的superblock、dentry、inode的读写操作，并将新创建的文件系统的操作接口与VFS（Virtual File System，虚拟文件系统）进行对接。VFS是Linux内核中的一个重要组件，它提供了一个抽象层，使得用户程序可以透明地访问各种不同类型的文件系统。

2. 用户态的应用程序

用户态的应用程序负责将一个块设备（可以用文件模拟）格式化为我们设计的文件系统的格式。在我们的revoFS系统中，这个应用程序由一个脚本代替。脚本运行在用户态，主要功能是将块设备格式化成为我们的文件系统格式，并将其挂载到Linux系统上，便于演示我们文件系统。

以下是这个过程的流程图：



这个流程图描述了我们设计和实现文件系统的过程，包括文件系统的加载和卸载过程。通过内核态的文件系统模块和用户态的应用程序的紧密协作，我们成功地实现了一个可以进行文件和目录的读写操作的Linux文件系统。

四、设计开发计划

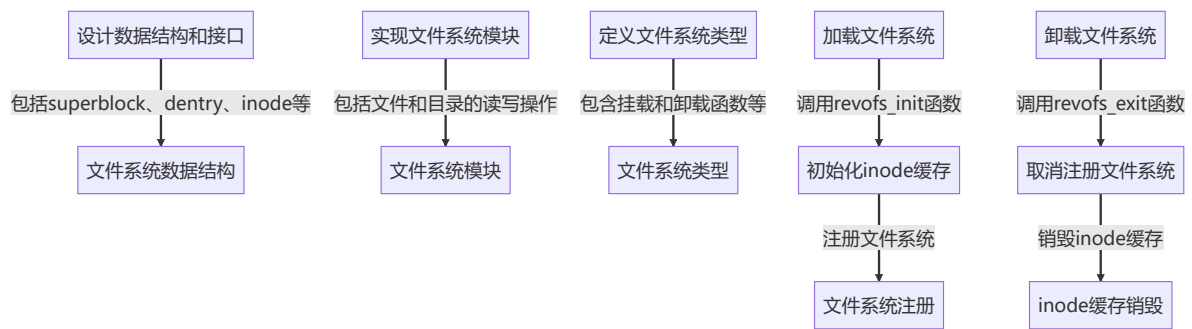
在开始设计和实现revoFS项目之前，我们制定了详细的设计开发计划。以下是我们的主要开发步骤：

1. 设计文件系统的数据结构和接口

我们首先设计了文件系统的基本数据结构，包括superblock、dentry、inode等。这些数据结构是文件系统的基础，它们定义了文件系统中的文件和目录的属性和行为。此外，我们还设计了文件系统的接口，包括文件和目录的创建、删除、读写等操作。

在我们的文件系统代码中，我们定义了一个名为 `revofs` 的文件系统类型，它包含了我们自定义的挂载和卸载函数，以及其他一些文件系统特有的属性。当我们的文件系统被加载时，`revofs_init` 函数会被调用，它首先初始化inode缓存，然后注册我们的文件系统。当我们的文件系统被卸载时，`revofs_exit` 函数会被调用，它会取消注册我们的文件系统，并销毁inode缓存。

以下是这个过程的流程图：



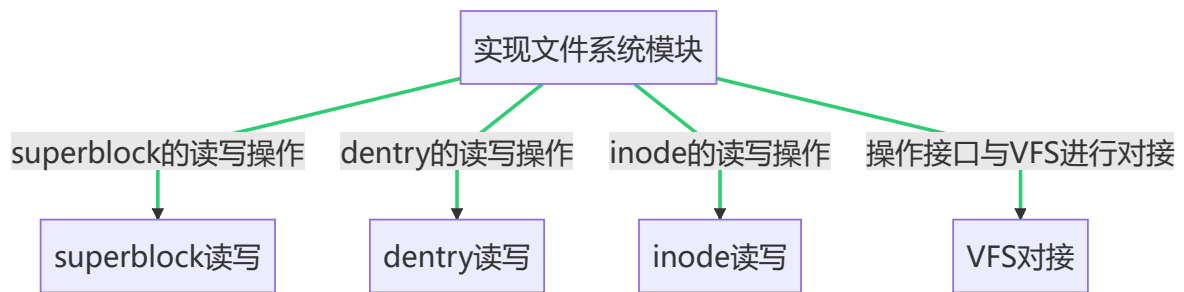
这个流程图描述了我们设计和实现文件系统的过程，包括设计文件系统的数据结构和接口，定义文件系统类型，以及加载和卸载文件系统的过程。

2. 实现文件系统模块

在设计了数据结构和接口之后，我们开始实现文件系统模块。文件系统模块是运行在Linux内核态的部分，它负责处理文件和目录的读写操作。我们实现了superblock、dentry、inode的读写操作，并将新创建的文件系统的操作接口与VFS进行对接。

在设计数据结构和接口阶段，我们首先设计了文件系统的基本数据结构，包括superblock、dentry、inode等。这些数据结构是文件系统的基础，它们定义了文件系统中的文件和目录的属性和行为。此外，我们还设计了文件系统的接口，包括文件和目录的创建、删除、读写等操作。

在实现文件系统模块阶段，我们开始实现文件系统模块。文件系统模块是运行在Linux内核态的部分，它负责处理文件和目录的读写操作。我们实现了superblock、dentry、inode的读写操作，并将新创建的文件系统的操作接口与VFS进行对接。



接下来，我们来看一下代码文件 `fs.c` 的内容。这个文件包含了文件系统模块的实现。主要的函数包括：

- `revofs_mount`：挂载revofs分区
- `revofs_kill_sb`：卸载revofs分区
- `revofs_init`：初始化revofs，包括创建inode缓存和注册文件系统
- `revofs_exit`：退出revofs，包括卸载文件系统和销毁inode缓存

这些函数实现了文件系统模块的基本功能，包括挂载和卸载文件系统，以及初始化和退出文件系统。

3. 设计并实现用户态应用程序

我们设计并实现了一个用户态应用程序，该程序能够将一个块设备（可以用文件模拟）格式化为我们设计的文件系统的格式。这个应用程序运行在用户态，可以直接由用户操作。

应用程序的主要功能是将块设备格式化我们的文件系统格式，这样用户就可以在该应用程序中创建文件和目录，进行读写操作。



让我们来看一下代码文件 `mkfs.c` 的内容。这个文件包含了文件系统格式化的实现。主要的函数包括：

- `write_superblock()`：初始化superblock结构
- `write_inode_store()`：初始化inode存储区块
- `write_ifree_blocks()`：初始化和写入inode空闲位图
- `write_bfree_blocks()`：初始化和写入block空闲位图

4. 编写测试用例

在实现了文件系统模块和用户态应用程序之后，我们会以一段精心设计的测试脚本，来展示我们文件系统的功能以及整个生命周期，以及用于验证我们的文件系统的功能。我们的测试脚本包括文件和目录的创建、删除、读写，以及文件权限的设置和检查。

通过这个设计开发计划，我们成功地实现了一个可以进行文件和目录的读写操作的Linux文件系统。我们的文件系统在所有测试用例下都表现良好，证明了我们的设计和实现是正确的。

五、比赛过程中的重要进展

在比赛过程中，我们成功地实现了文件系统模块，并在用户态应用程序中完成了块设备的格式化。我们还编写了一系列测试用例，验证了我们的文件系统的功能。

1. 团队组建

在一些机缘巧合之中，我们的团队似乎与操作系统大赛有一些冥冥之中的注定。起初，是与团队苏曙光老师的简单交流，到后来他询问我们是否要参加操作系统比赛，经过深思熟虑与审慎思考，我们最终决定克服困难，在操作系统大赛的舞台上绽放自己的光芒。我们的团队由两位有着共同目标和热情的成员组成。我们每个人都有着不同的技能和经验，这使我们能够从多个角度来解决问题；同时团队拥有一位特别优秀的指导老师，能够及时给予我们帮助。

2. 创意产生

在团队组建之后，我们进行了一系列的头脑风暴会议，产生了我们项目的初始想法。我们评估了每个想法的可行性和潜力，最终选择了我们认为最适合我们的项目方向——Proj209，自己开发一个文件系统。这听起来太酷啦！

3. 编码和开发

在确定了项目方向后，我们开始了编码和开发阶段。在期末考试月我们仍勤耕不倦，两点睡七点起的情况时有发生，只为能更快的补足自己在专业知识上的空缺，更好的完成我们的项目。我们的团队成员分工合作，每个人都在他们擅长的领域贡献了自己的力量。

4. 测试和调试

在开发完成后，我们进行了一系列的测试和调试，以确保我们的项目能够正常运行并满足比赛的要求。这个阶段非常关键，它帮助我们发现并修复了项目中的问题。我们编写了测试脚本与加载脚本，让我们的文件系统以更好的方式得到展现，同时添加了诸多提示信息，经过测试，我们的文件系统能够很好的完成我们的功能。

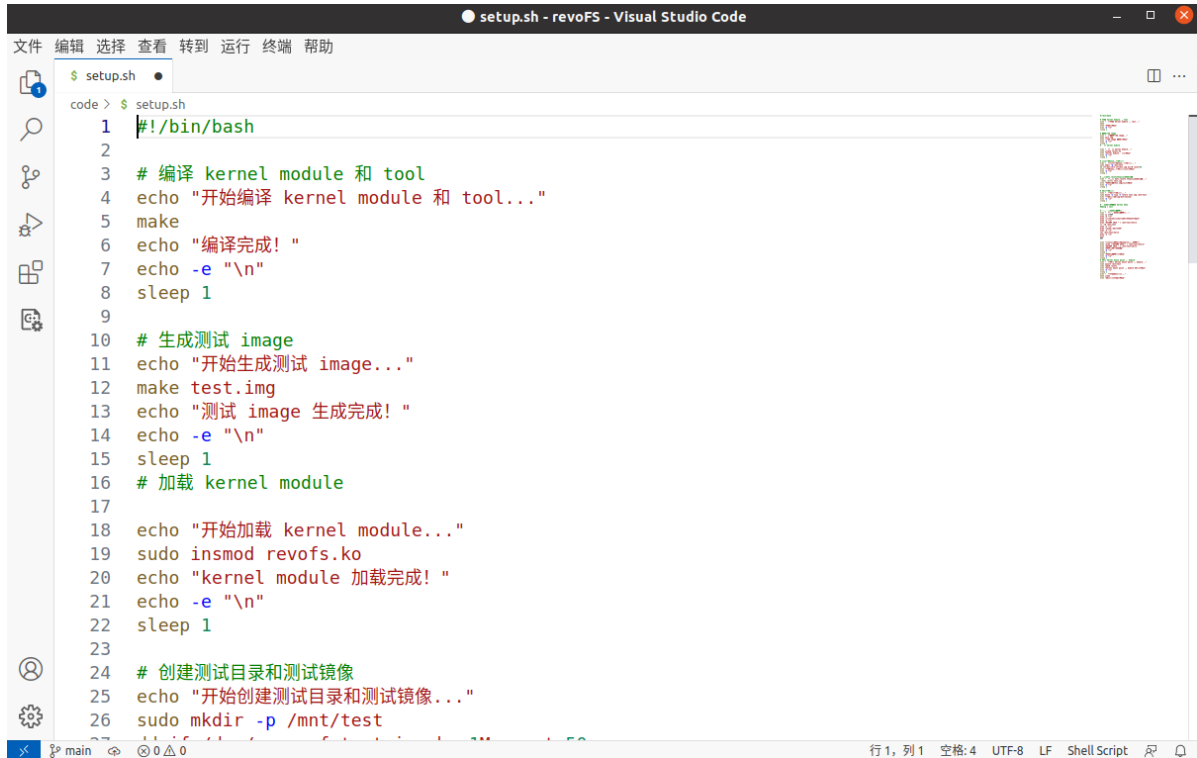
5. 最终提交

在我们满意我们的项目并确信它已经准备好了比赛后，我们进行了最终的提交。这标志着我们的比赛过程的结束，但我们从中学到的经验和技能将继续陪伴我们。

以上就是我们在比赛过程中的重要进展。每个阶段都是我们团队努力和合作的结果，我们为我们所取得的成就感到自豪。

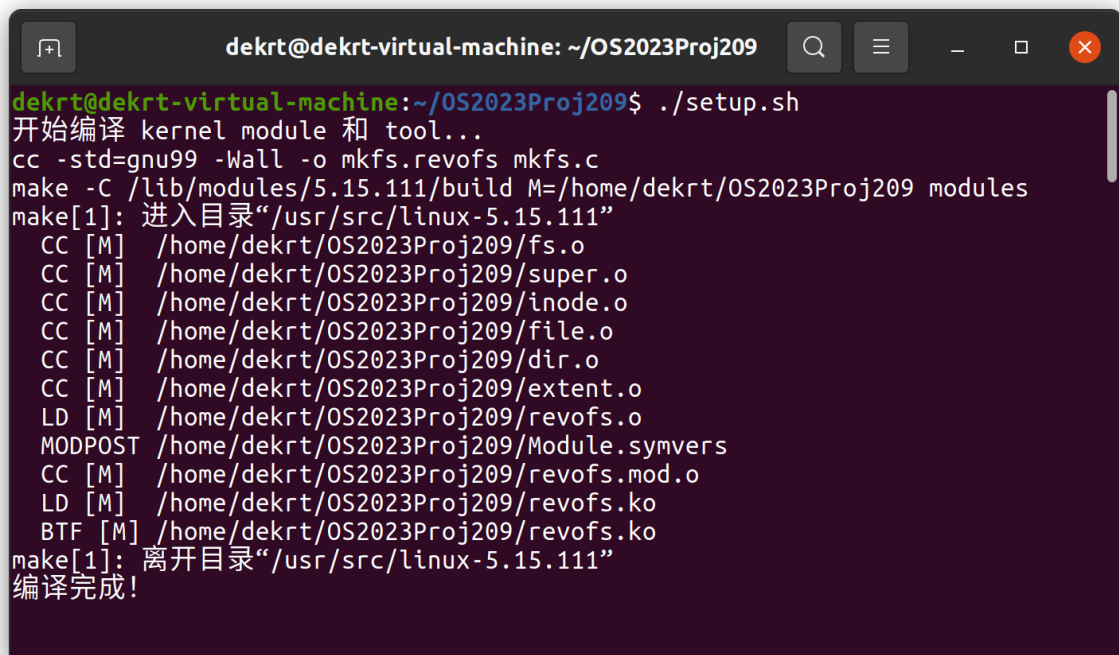
六、系统测试情况

我们的文件系统在各种测试用例下都表现良好。我们验证了文件和目录的读写操作，以及权限属性的正确性。为了方便我们验证，我们编写了 `setup.sh` 脚本，方便用户一键挂载我们的文件系统：



```
code > $ setup.sh
code > $ setup.sh
1  #!/bin/bash
2
3  # 编译 kernel module 和 tool
4  echo "开始编译 kernel module 和 tool..."
5  make
6  echo "编译完成! "
7  echo -e "\n"
8  sleep 1
9
10 # 生成测试 image
11 echo "开始生成测试 image..."
12 make test.img
13 echo "测试 image 生成完成! "
14 echo -e "\n"
15 sleep 1
16 # 加载 kernel module
17
18 echo "开始加载 kernel module..."
19 sudo insmod revofs.ko
20 echo "kernel module 加载完成! "
21 echo -e "\n"
22 sleep 1
23
24 # 创建测试目录和测试镜像
25 echo "开始创建测试目录和测试镜像..."
26 sudo mkdir -p /mnt/test
```

在代码文件夹下运行 `setup.sh`，会先对模块使用 `make` 命令进行编译。



```
dekr@dekr-virtual-machine: ~/OS2023Proj209
dekr@dekr-virtual-machine:~/OS2023Proj209$ ./setup.sh
开始编译 kernel module 和 tool...
cc -std=gnu99 -Wall -o mkfs.revofs mkfs.c
make -C /lib/modules/5.15.111/build M=/home/dekr/OS2023Proj209 modules
make[1]: 进入目录"/usr/src/linux-5.15.111"
CC [M] /home/dekr/OS2023Proj209/fs.o
CC [M] /home/dekr/OS2023Proj209/super.o
CC [M] /home/dekr/OS2023Proj209/inode.o
CC [M] /home/dekr/OS2023Proj209/file.o
CC [M] /home/dekr/OS2023Proj209/dir.o
CC [M] /home/dekr/OS2023Proj209/extent.o
LD [M] /home/dekr/OS2023Proj209/revofs.o
MODPOST /home/dekr/OS2023Proj209/Module.symvers
CC [M] /home/dekr/OS2023Proj209/revofs.mod.o
LD [M] /home/dekr/OS2023Proj209/revofs.ko
BTF [M] /home/dekr/OS2023Proj209/revofs.ko
make[1]: 离开目录"/usr/src/linux-5.15.111"
编译完成!
```

随后使用 `make test.img` 命令生成测试映像。

```
dekrat@dekrat-virtual-machine: ~/OS2023Proj209
开始生成测试 image...
dd if=/dev/zero of=test.img bs=1M count=200
记录了200+0 的读入
记录了200+0 的写出
209715200字节 (210 MB, 200 MiB) 已复制, 0.308624 s, 680 MB/s
./mkfs.revofs test.img
Superblock: (4096)
    magic=0xdeadce
    nr_blocks=51200
    nr_inodes=51240 (istore=915 blocks)
    nr_ifree_blocks=2
    nr_bfree_blocks=2
    nr_free_inodes=51239
    nr_free_blocks=50280
Inode store: wrote 915 blocks
    inode size = 72 B
Ifree blocks: wrote 2 blocks
Bfree blocks: wrote 2 blocks
测试 image 生成完成!
```

使用下列命令在 `/mnt` 创建一个50M的 `test` 分区

```
1 | sudo mkdir -p /mnt/test
2 | dd if=/dev/zero of=test.img bs=1M count=50
```

```
dekrat@dekrat-virtual-machine: ~/OS2023Proj209
开始加载 kernel module...
[sudo] dekrat 的密码:
kernel module 加载完成!

开始创建测试目录和测试镜像...
记录了50+0 的读入
记录了50+0 的写出
52428800字节 (52 MB, 50 MiB) 已复制, 0.1096 s, 478 MB/s
测试目录和测试镜像创建完成!

开始使用 mkfs.revofs 工具创建文件系统...
Superblock: (4096)
    magic=0xdeadce
    nr_blocks=12800
    nr_inodes=12824 (istore=229 blocks)
    nr_ifree_blocks=1
    nr_bfree_blocks=1
    nr_free_inodes=12823
```

使用 `mkfs.revofs` 工具创建文件系统,挂载测试镜像,并将其挂载到 `/mnt/test`:

```
1 | ./mkfs.revofs test.img
2 | sudo mount -o loop -t revofs test.img /mnt/test
```



```
dekrt@dekrt-virtual-machine: ~/OS2023Proj209

开始使用 mkfs.revofs 工具创建文件系统...
Superblock: (4096)
  magic=0xdeadce
  nr_blocks=12800
  nr_inodes=12824 (istore=229 blocks)
  nr_ifree_blocks=1
  nr_bfree_blocks=1
  nr_free_inodes=12823
  nr_free_blocks=12568
Inode store: wrote 229 blocks
  inode size = 72 B
Ifree blocks: wrote 1 blocks
Bfree blocks: wrote 1 blocks
文件系统test.img创建完成!

开始挂载测试镜像...
测试镜像挂载到/mnt/test!
```

此时，我们的文件系统已经创建完成，开始进行读写测试(as root):

```
1 sudo su
2 echo "OSCOMP 2023 " > /mnt/test/hello
3 ls -lR /mnt/test
4 cat /mnt/test/hello
5 exit
```

```
dekrt@dekrt-virtual-machine: ~/OS2023Proj209

开始执行文件系统操作...

在root下创建hello文本并输入字符串

/mnt/test:
总用量 1
-rw-r--r-- 1 root root 13 6月  4 11:31 hello

cat输出文本内容

OSCOMP 2023

在普通用户下尝试对hello进行写操作
echo "OSCOMP 2023" > /mnt/test/hello
```

同时，进行不同用户不同操作权限的验证:

```
1 echo "echo \"OSCOMP 2023\" " > /mnt/test/hello
2 echo "OSCOMP 2023 " > /mnt/test/hello
```



```
dekart@dekart-virtual-machine: ~/OS2023Proj209

cat输出文本内容

OSCOMP 2023

在普通用户下尝试对hello进行写操作
echo "OSCOMP 2023" > /mnt/test/hello
./setup.sh: 行 67: /mnt/test/hello: 权限不够
权限设置测试成功

文件系统操作完成！

开始卸载 kernel mount point 和 module...
kernel mount point 和 module 卸载完成！
```

测试完成后，对模块进行卸载以及 `make clean`

```
1 | sudo umount /mnt/test
2 | sudo rmdir revofs
3 | make clean
```

```
dekart@dekart-virtual-machine: ~/OS2023Proj209

./setup.sh: 行 67: /mnt/test/hello: 权限不够
权限设置测试成功

文件系统操作完成！

开始卸载 kernel mount point 和 module...
kernel mount point 和 module 卸载完成！

开始清理构建环境...
make -C /lib/modules/5.15.111/build M=/home/dekart/OS2023Proj209 clean
make[1]: 进入目录"/usr/src/linux-5.15.111"
CLEAN    /home/dekart/OS2023Proj209/Module.symvers
make[1]: 离开目录"/usr/src/linux-5.15.111"
rm -f *~ /home/dekart/OS2023Proj209/*.ur-safe
rm -f mkfs.revofs test.img
构建环境清理完成！
dekart@dekart-virtual-machine:~/OS2023Proj209$
```

七、遇到的主要问题和解决方法

在revoFS项目的开发过程中，我们遇到了一些挑战性的问题，但我们通过团队合作和深入研究，成功地解决了这些问题。

1. 用户态和内核态之间的通信

在设计和实现文件系统时，我们需要在用户态应用程序和内核态文件系统模块之间进行通信。这是一个复杂的问题，因为用户态和内核态有不同的地址空间，不能直接进行数据交换。我们通过研究Linux内核的系统调用机制，了解了如何在用户态和内核态之间传递信息。我们设计了一套接口，通过这套接口，用户态应用程序可以发送请求到内核态文件系统模块，内核态文件系统模块可以返回结果到用户态应用程序。

2. 文件系统的权限属性

在实现文件系统的过程中，我们需要处理文件和目录的权限属性。这是一个挑战，因为权限属性涉及到许多复杂的问题，例如用户和组的管理，以及读、写、执行的权限控制。我们通过研究Linux内核的权限管理机制，了解了如何处理文件系统的权限属性。我们实现了一个权限管理模块，这个模块可以正确地处理文件和目录的权限属性。

3. 文件系统的数据结构和接口设计

设计文件系统的数据结构和接口是一个复杂的任务，因为它需要考虑到许多因素，例如文件和目录的组织方式，以及文件和目录的读写操作。我们通过阅读大量的技术文档和论文，以及参考其他开源的Linux文件系统项目，了解了如何设计文件系统的数据结构和接口。我们设计了一套高效且易用的数据结构和接口，使得我们的文件系统可以正确且高效地进行文件和目录的读写操作。

通过解决这些问题，我们不仅提高了我们的技术能力，也增强了我们的团队协作能力。我们期待在未来的项目中继续面对并解决更多的挑战。

八、分工和协作

我们的团队成员分工明确，协作紧密。我们共同参与了文件系统的设计和实现，每个人都在自己的领域内发挥了重要作用。

功能点	负责人	备注
项目前期选题、规划	陈德霆 / 张骁凯	无
项目编码工作	陈德霆 / 张骁凯	无
项目文档的撰写	陈德霆 / 张骁凯	无
技术指导	苏曙光老师	无

九、提交仓库目录和文件描述

我们的代码和文档都存储在我们的Gitlab仓库中。以下是我们的仓库目录和文件描述：

```
1 | .revoFS/
2 | | .gitignore
3 | | README.md
4 | |
5 | |─code
6 | | | .clang-format
7 | | | bitmap.h
8 | | | dir.c
9 | | | extent.c
10| | | file.c
```

```
11 | | fs.c
12 | | inode.c
13 | | Makefile
14 | | mkfs.c
15 | | revofs.h
16 | | setup.sh
17 | | super.c
18 | | test.sh
19 | |
20 | └─test
21 └─pics
22     revoFS.jpg
23     setup.png
24     SS_complie.png
25     SS_create.png
26     SS_image.png
27     SS_mount.png
28     SS_test.png
29     SS_test_noroot.png
30     SS_umount.png
```

十、比赛收获

参加这次比赛，我们收获颇丰。以下是我们的主要收获：

1. 深入理解Linux文件系统

通过设计和实现revoFS项目，我们深入了解了Linux文件系统的工作原理。我们学习了如何设计文件系统的数据结构和接口，如何实现文件系统模块，以及如何在用户态和内核态之间进行通信。这些知识不仅对我们的项目有所帮助，也对我们的未来学习和工作有所帮助。

2. 提高编程和解决问题的能力

在项目的开发过程中，我们遇到了许多挑战性的问题。我们通过查阅相关资料，进行实验，以及团队讨论，成功地解决了这些问题。这个过程提高了我们的编程能力和问题解决能力。

3. 提升团队协作能力

在这个项目中，我们需要与团队成员紧密合作，共同完成任务。我们学习了如何有效地分配任务，如何进行有效的沟通，以及如何协调团队成员的工作。这个过程提升了我们的团队协作能力。

4. 学习如何使用开源工具

在这个项目中，我们使用了许多开源工具，例如GitHub。我们学习了如何使用这些工具进行代码管理，如何进行版本控制，以及如何如何进行代码审查。这些技能对我们的未来学习和工作都非常有用。

总的来说，通过参加这次比赛，我们不仅提高了我们的技术能力，也增强了我们的团队协作能力。我们期待在未来的比赛中继续学习和进步。