



# 实验五：基于FUSE的青春版EXT2文件系统

《操作系统》课程实验

# ● 目录

---



## ● 实验目的

---

- 以Linux系统中的EXT2文件系统为例，熟悉该文件系统内部数据结构的组织方式和基本处理流程；
- 基于 FUSE 设计并实现一个可以真正在Linux上跑 的文件系统。

## ● 实验内容（一）

熟悉DDRIVER驱动，根据提示修改其中的一处错误：

```
~/user-land-filesystem/tests/test_ddriver$ cmake CMakeLists.txt
```

```
~/user-land-filesystem/tests/test_ddriver$ make
```

```
~/user-land-filesystem/tests/test_ddriver$ ./ddriver_test
```

```
[cs@localhost test_ddriver]$ make
Scanning dependencies of target ddriver_test
[ 50%] Building C object CMakeFiles/ddriver_test.dir/src/test.c.o
[100%] Linking C executable ddriver_test
[100%] Built target ddriver_test
[cs@localhost test_ddriver]$ ./ddriver_test
PANIC: wrong path [/root/ddriver], should be [/home/cs/ddriver]
[cs@localhost test_ddriver]$
```

## ● 实验内容（二）

---

### 基于FUSE实现青春版EXT2文件系统：

系统结构参考EXT2文件系统，要求具有SUPER\_BLOCK、DATA\_MAP、INODE\_MAP等主要结构。

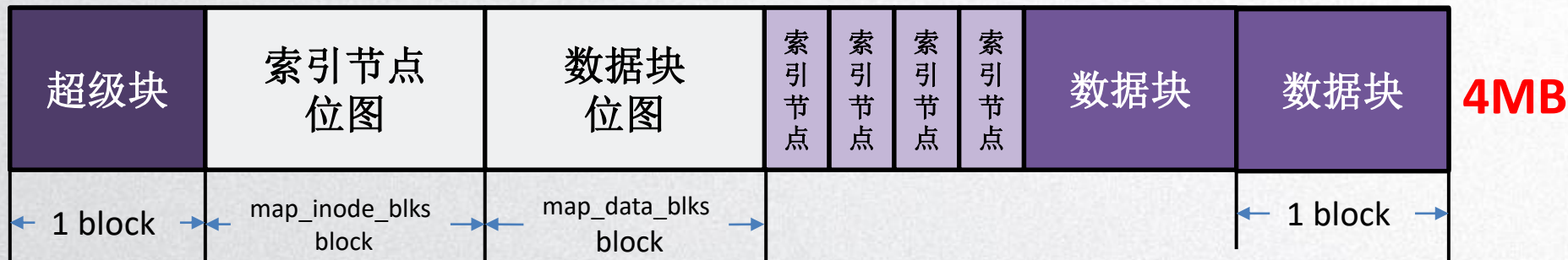
- 挂载文件系统， mount
- 卸载文件系统， umount
- 创建文件/文件夹， touch/mkdir
- 查看文件夹下的文件， ls

**注：**以上未提及功能（如复制文件夹、删除文件、读写文件等）不要求实现。



## ● 实验原理 | 青春版（简化的）Ext2文件系统

- Ext2文件系统将盘块分成两大类：保存元数据（管理数据）的元数据盘块，以及存放文件内容数据的数据盘块。



- ① **超级块**：包含整个系统的总体信息；
- ② **索引节点位图**：记录着索引节点表的使用情况，用1个比特记录某一个索引节点是否被使用；
- ③ **数据块位图**：记录着数据块的使用情况，用1个比特记录某一个数据块是否被占用；
- ④ **索引节点 (inode)**：记录着文件的元数据，每个文件都与一个inode对应。
- ⑤ **数据块**：记录文件内容，若文件太大时，会占用多个 block。

## ● 实验原理 | 超级块superblock

- 超级块是文件系统的起点系统，记录了文件系统全局性的一些信息。

下面给出一个参考的超级块结构体示例：

```
struct super_block_d {  
    uint32_t magic_num;    // 幻数  
    int      max_ino;      // 最多支持的文件数  
  
    int      map_inode_blks; // inode位图占用的块数  
    int      map_inode_offset; // inode位图在磁盘上的偏移  
  
    int      map_data_blks; // data位图占用的块数  
    int      map_data_offset; // data位图在磁盘上的偏移  
};
```

### 如何决定位图大小呢？

- ① 假设估算。假定每个文件具有固定大小，从而估算整个文件系统可以管理多少文件，从而计算位图大小。
- ② 规定大小。为了简单起见，我们可以自行规定位图的大小，只要保证指令运行成功即可。

## ● 实验原理 | 数据块data block

---

- data block 是用来放置文件内容数据地方, block 大小有 1KB
- 数据块特性:
  - block 的大小与数量在格式化完就不能够再改变了(除非重新格式化);
  - 每个 block 内最多只能够放置一个文件的数据;
  - 如果文件大于 block 的大小, 则一个文件会占用多个 block 数量;
  - 若文件小于 block , 则该 block 的剩余容量就不能够再被使用了(磁盘空间会浪费)。



## ● 实验原理 | Inode数据区

### ➤ Inode索引节点特性:

- **你需要自己设计inode**
- 每个文件都仅会占用一个inode
- 文件系统能够创建的文件数量与 inode 的数量有关
- 读文件时先找到inode，再读取block内容。

inode 能够指定多大的文件呢？

直接索引方式：一个文件不能超过

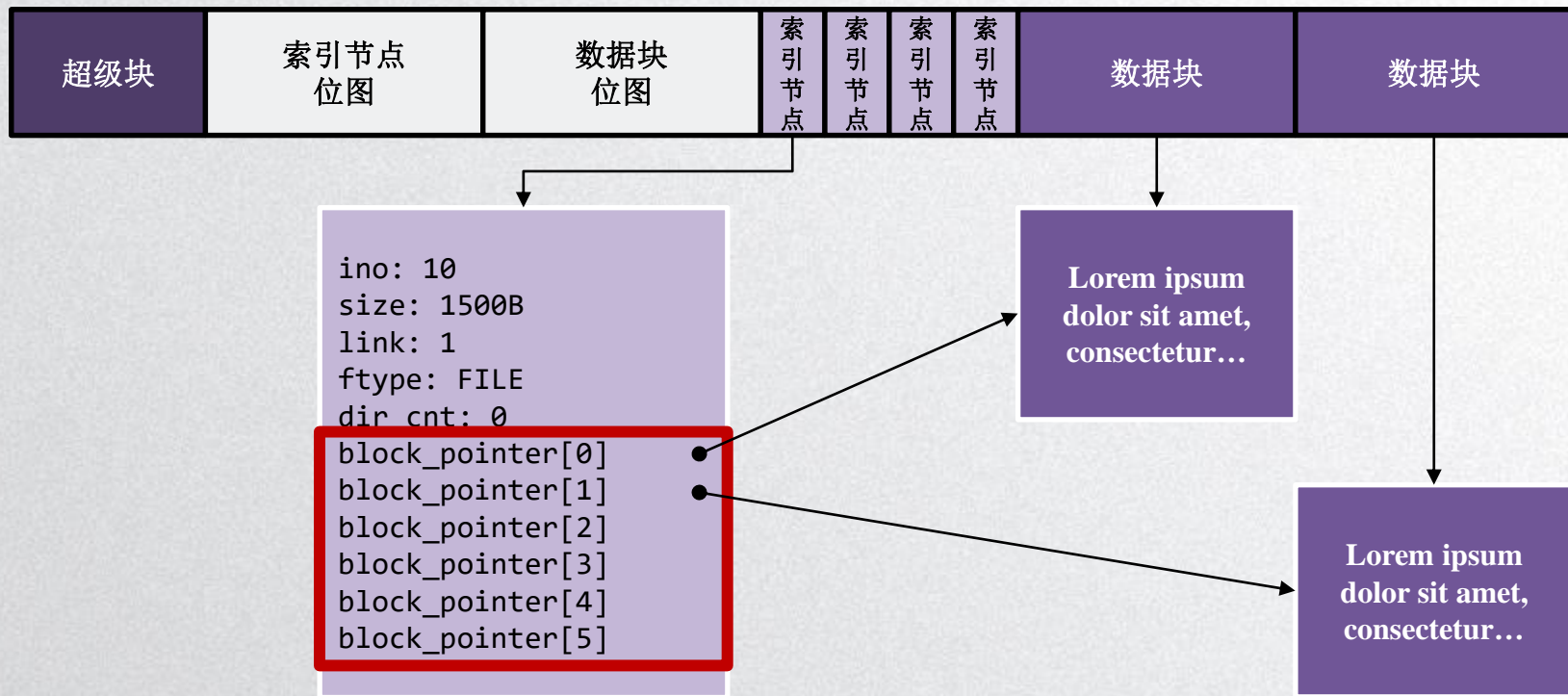
6个数据块 \* 1KB数据块大小 = 6KB

```
struct inode_d {  
    int ino; // 在inode位图中的下标  
    int size; // 文件已占用空间  
    int link; // 链接数  
    FILE_TYPE ftype; // 文件类型（目录类型、普通文件类型）  
    int dir_cnt; // 如果是目录类型文件，下面有几个目录项  
    int block_pointer[6]; // 数据块指针（可固定分配）  
};
```

参考示例

## ● 实验原理 | Inode数据区

- inode里保存了相应数据块的索引值。
- 读写文件时则通过索引值找到相应的数据块来读写。



## ● 实验原理 | 文件夹（目录）

---

- 文件夹（目录）用于将整个文件系统的文件行程按路径名访问的**树形组织结构**排列；
- Ext2文件系统并没有单独形成用于目录数据的数据块，而是将目录与普通文件一样**存放在数据块**中；
- 目录数据块中的内容是目录项的数组，通过**上一级目录项**将其类型标识为“目录”以区分普通文件；
- **根目录有固定的起点**（例如Ext2文件系统根目录的索引号为2），因此根目录文件就不需要上级目录来定位了。

## ● 实验原理 | 文件夹（目录）

### ➤ 文件夹特性：

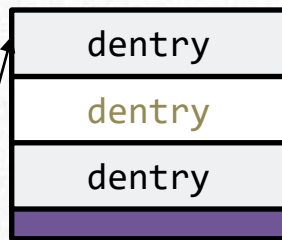
- 你需要自己设计dentry\_d

1个block可以存储 ? 个dentry

```
struct dentry_d {  
    char fname[MAX_FILE_NAME]; // 指向的ino文件名  
    FILE_TYPE ftype; // 指向的ino文件类型  
    int ino; // 指向的ino号  
    int valid; // 该目录项是否有效  
};
```

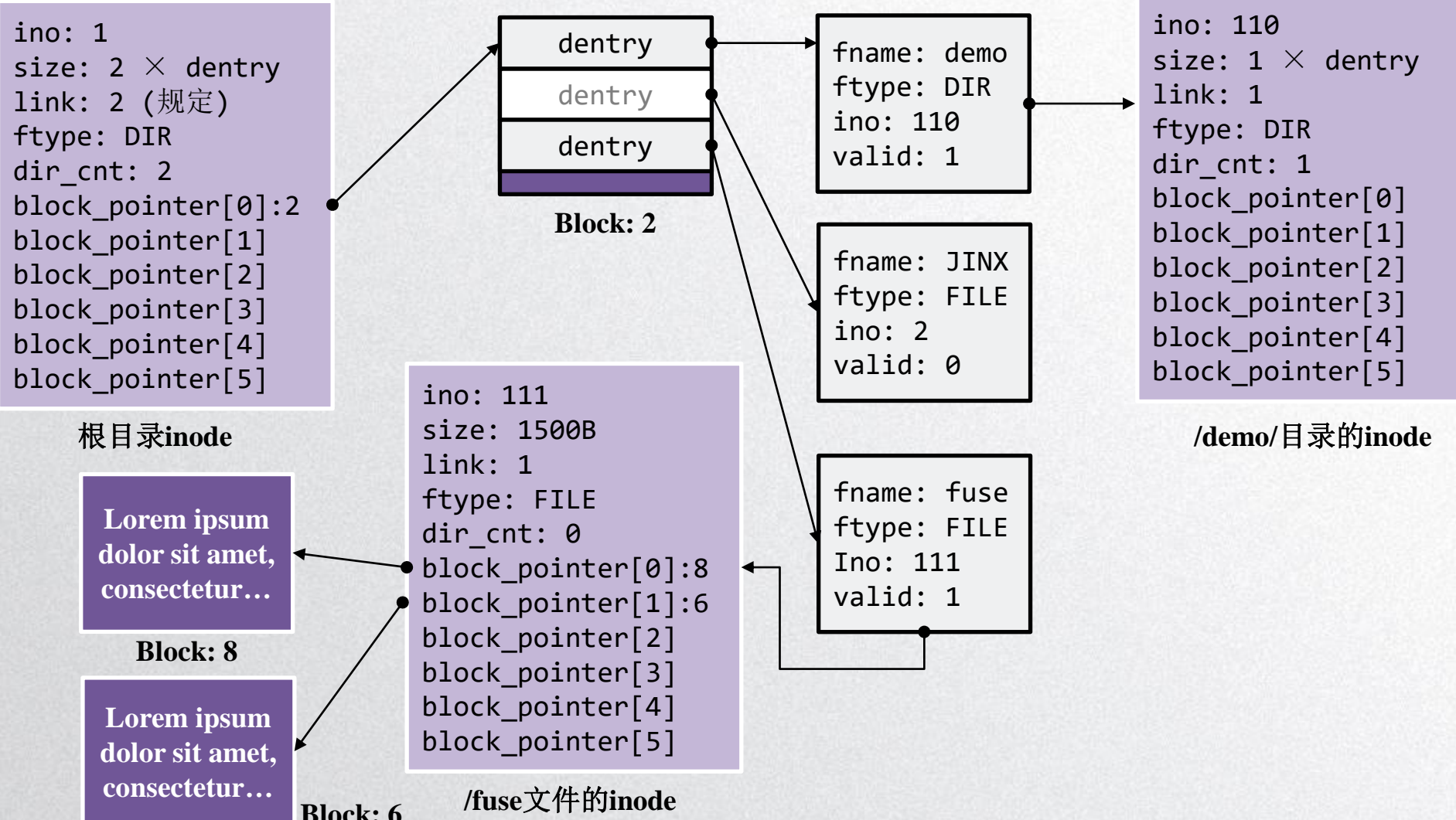
```
ino: 1  
size: 2 × dentry  
link: 2 (规定)  
ftype: DIR  
dir_cnt: 2  
block_pointer[0]: 2  
block_pointer[1]  
block_pointer[2]  
block_pointer[3]  
block_pointer[4]  
block_pointer[5]
```

根目录inode



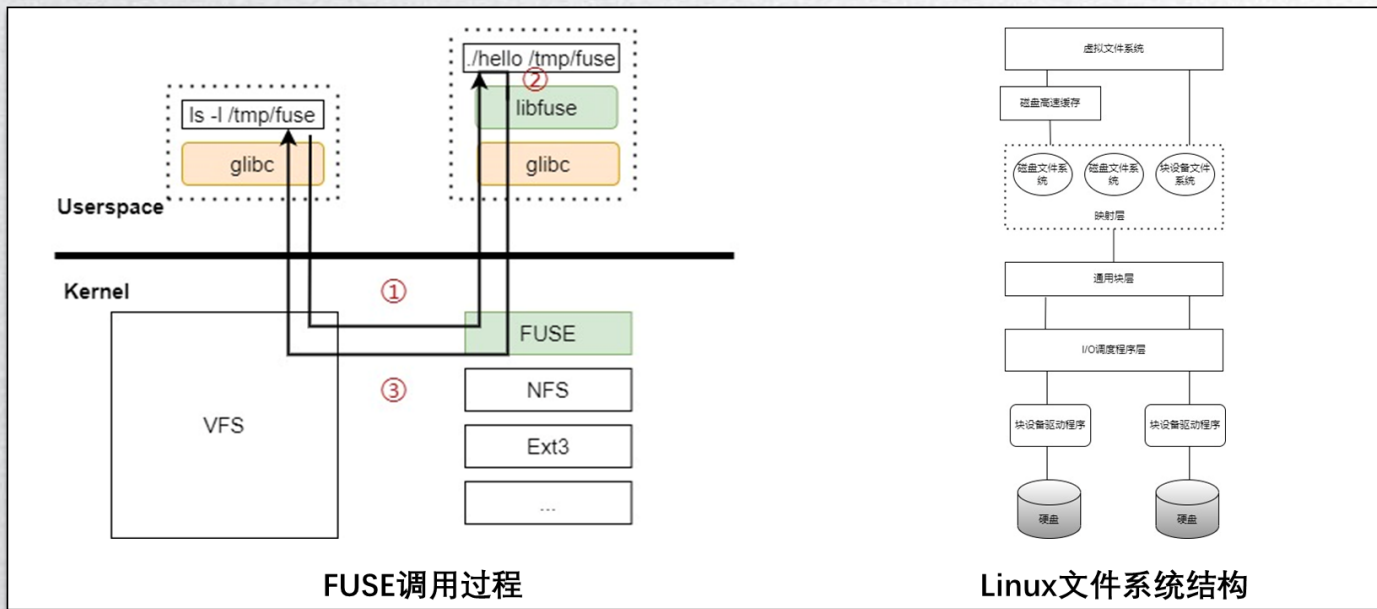
Block: 2





## ● 实验原理 | FUSE架构

- FUSE (Filesystem in User Space) 将文件系统的实现从内核态搬到了用户态。
  - **Step ①**: 如果在该目录中有相关操作（如调用`ls`命令`）时，请求会经过Linux虚拟文件系统VFS层 到FUSE的内核模块；
  - **Step ②**: FUSE内核模块根据请求类型，调用用户态应用注册的函数；
  - **Step ③**: 将处理结果通过VFS返回给系统调用。



## ● 实验步骤 |

---

### 任务一：熟悉DDRIVER驱动

- **Step 1**: 配置开发环境，按照指导书给的步骤来建立环境
- **Step 2**: 要求熟悉DDRIVER驱动，我们要求大家根据注释看懂user-land-filesystem/tests/test\_ddriver/src/test.c的代码，并根据提示修改其中的一处错误

```
[cs@localhost test_ddriver]$ make
Scanning dependencies of target ddriver_test
[ 50%] Building C object CMakeFiles/ddriver_test.dir/src/test.c.o
[100%] Linking C executable ddriver_test
[100%] Built target ddriver_test
[cs@localhost test_ddriver]$ ./ddriver_test
PANIC: wrong path [/root/ddriver], should be [/home/cs/ddriver]
[cs@localhost test_ddriver]$
```

# ● 实验步骤 |

## 任务二：基于FUSE实现青春版EXT2文件系统

可参考两个FUSE文件系统：

### ➤ simplefs (fs/simplefs文件夹下)

simplefs是一个类EXT2的文件系统，

但没有给予数据位图的实现；

### ➤ myfs (fs/samples文件夹下) ,

myfs是github上的一个开源项目，

也是一个FUSE文件系统实例。

- 封装对 `ddriver` 的访问代码，方便设备读写。注意驱动读写IO为512B，但是EXT2文件系统一个块大小为1024B，也就是两个IO单位。在simplefs文件系统中也封装了 `ddriver` 的访问代码，然而simplefs是按一个块大小为512B来封装的（详见 `sfs_utils.c` 的 `sfs_driver_read` 函数）。
- 设计介质数据结构（位于 `types.h`）；
- 设计内存数据结构（可设计、可不设计）；
- 仔细阅读 `include/fs.layout` 文件，并按要求设计填写自己的文件系统布局，在测试 `umount` 时会
- 对文件系统布局进行检测；
- 完成 `.init` 钩子：读 超级块、初始化文件系统；
- 完成 `.destroy` 钩子：回写必要结构，保证下一次挂载正常；
- 验证挂载卸载流程的正确性（主要看读写是否正确）；
- 完成工具函数（可自行设计）：
  - 完成 分配inode 函数：创建一个Inode、初始化其上的字段、修改位图结构，可能还需要修改超级块；
  - 完成 分配dentry 函数：创建一个dentry、初始化其上的字段；
  - 完成将 dentry加入到inode 中的函数：将生成的dentry写入到inode中；
  - 完成 路径解析函数，要么返回dentry，要么返回inode，可自行设计；
- 根据工具函数，完成 `.get_attr` 钩子；
- 根据工具函数，完成 `.mknod` 钩子；
- 根据工具函数，完成 `.mkdir` 钩子；
- 根据工具函数，完成 `.readdir` 钩子；
- 手工测试 `touch`、`ls`、`mkdir`、`fusermount -u` 等命令的正确性；
- 通过 `./tests/test.sh` 脚本的基本功能测试。

关于环境配置、sfs简介及测试介绍请到bilibili上观看：  
[https://www.bilibili.com/video/BV1KB4y1z7f2/?spm\\_id\\_from=333.999.0.0](https://www.bilibili.com/video/BV1KB4y1z7f2/?spm_id_from=333.999.0.0)

F-Tutorials: HITSZ FUSE文件系统实验

164 0 2022-08-18 19:42:05 未经作者授权，禁止转载

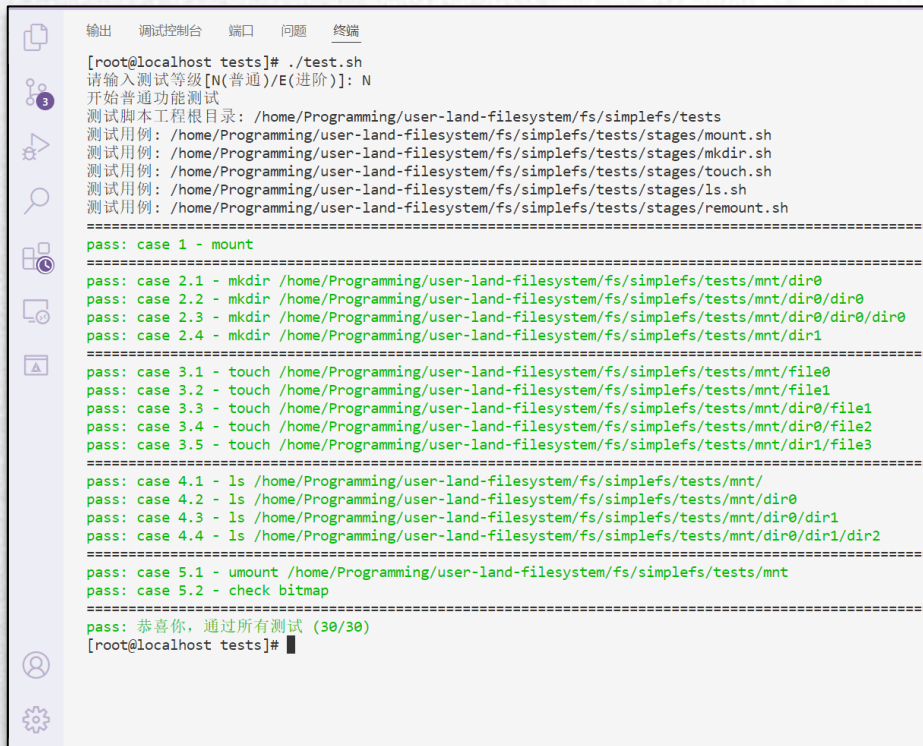




# 实验步骤 |

## 实验评测

- **基础测试**: mount, mkdir, touch, ls, umount测试
- **进阶测试**: mount, mkdir, touch, ls, read&write, cp, umount测试（功能进阶）
- **分点测试**: 对于各个功能点单独测试



```
[root@localhost tests]# ./test.sh
请输入测试等级 [N(普通)/E(进阶)]: N
开始普通功能测试
测试脚本工程根目录: /home/Programming/user-land-filesystem/fs/simplefs/tests
测试用例: /home/Programming/user-land-filesystem/fs/simplefs/tests/stages/mount.sh
测试用例: /home/Programming/user-land-filesystem/fs/simplefs/tests/stages/mkdir.sh
测试用例: /home/Programming/user-land-filesystem/fs/simplefs/tests/stages/touch.sh
测试用例: /home/Programming/user-land-filesystem/fs/simplefs/tests/stages/ls.sh
测试用例: /home/Programming/user-land-filesystem/fs/simplefs/tests/stages/remount.sh
=====
pass: case 1 - mount
=====
pass: case 2.1 - mkdir /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0
pass: case 2.2 - mkdir /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/dir0
pass: case 2.3 - mkdir /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/dir0/dir0
pass: case 2.4 - mkdir /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir1
=====
pass: case 3.1 - touch /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/file0
pass: case 3.2 - touch /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/file1
pass: case 3.3 - touch /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/file1
pass: case 3.4 - touch /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/file2
pass: case 3.5 - touch /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir1/file3
=====
pass: case 4.1 - ls /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/
pass: case 4.2 - ls /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0
pass: case 4.3 - ls /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/dir1
pass: case 4.4 - ls /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt/dir0/dir1/dir2
=====
pass: case 5.1 - umount /home/Programming/user-land-filesystem/fs/simplefs/tests/mnt
pass: case 5.2 - check bitmap
=====
pass: 恭喜你, 通过所有测试 (30/30)
[root@localhost tests]#
```

## ● 实验要点 | ① FUSE架构

- FUSE框架通过向fuse\_main函数传入一个被复制的fuse\_operations结构体即可完成操作的注册。
- 只要求同学们实现钩子函数（其中标记为 NULL 的在本次实验中不要求实现）

```
273  /**
274  * SECTION: FUSE入口
275  */
276  int main(int argc, char **argv)
277  {
278      int ret;
279      struct fuse_args args = FUSE_ARGS_INIT(argc, argv);
280
281      newfs_options.device = strdup("TODO: 这里填写你的ddriver设备路径");
282
283      if (fuse_opt_parse(&args, &newfs_options, option_spec, NULL) == -1)
284          return -1;
285
286      ret = fuse_main(args.argc, args.argv, &operations, NULL);
287      fuse_opt_free_args(&args);
288      return ret;
289  }
```

```
18  /**
19  * SECTION: FUSE操作定义
20  */
21  static struct fuse_operations operations = {
22      .init = newfs_init,           /* mount文件系统 */
23      .destroy = newfs_destroy,     /* umount文件系统 */
24      .mkdir = newfs_mkdir,         /* 建目录, mkdir */
25      .getattr = newfs_getattr,     /* 获取文件属性, 类似stat, 必须完成 */
26      .readdir = newfs_readdir,    /* 填充dentries */
27      .mknod = newfs_mknod,        /* 创建文件, touch相关 */
28      .write = NULL,               /* 写入文件 */
29      .read = NULL,                /* 读文件 */
30      .utimens = newfs_utimens,     /* 修改时间, 忽略, 避免touch报错 */
31      .truncate = NULL,            /* 改变文件大小 */
32      .unlink = NULL,              /* 删除文件 */
33      .rmdir = NULL,               /* 删除目录, rm -r */
34      .rename = NULL,              /* 重命名, mv */
35
36      .open = NULL,
37      .opendir = NULL,
38      .access = NULL
39  };
```

## ● 实验要点 | ②内存结构表示

- 建议同学们针对文件系统结构布局设计相应的内存数据结构来方便管理。
  - 以超级块为例，根据super\_block我们就能方便地对inode位图和data位图进行管理，并能够通过root\_dentry轻松访问根目录，从而达到快速解析路径的目的。

```
struct super_block_d
{
    uint32_t      magic_num;

    int           max_ino;

    int           map_inode_blks;
    int           map_inode_offset;

    int           map_data_blks;
    int           map_data_offset;
};
```

超级块的磁盘布局

```
struct super_block
{
    int           max_ino;

    uint8_t*      map_inode;      // inode位图
    uint8_t*      map_data;      // data位图

    struct dentry* root_dentry;   // 根目录dentry
};
```

超级块的内存数据结构

## ● 实现要点 | ③虚拟磁盘接口

- 接口模拟了对一个容量为4MB的磁盘的操作;
- 保留了磁盘的访问特性, 即按块访问, 每次读写都需要读写完整的 512B 的磁盘块, 若更新部分数据时, 需要读取整个磁盘块, 更新相应的内容, 再写回去;
- 驱动文件位于driver/user\_ddriver下, 编译方式是静态链接库, 无法直接通过Debug调试进入到上述接口实现的代码中, 这也是为了模拟真实驱动所做的考虑。

```
int ddriver_open(char *path);  
int ddriver_seek(int fd, off_t offset, int whence);  
int ddriver_write(int fd, char *buf, size_t size);  
int ddriver_read(int fd, char *buf, size_t size);  
int ddriver_ioctl(int fd, unsigned long cmd, void *ret);  
int ddriver_close(int fd);
```



## ● 实现要点 | ④注意事项

---

- 实验指导书 <https://hitsz-cslab.gitee.io/os-labs/lab5/part1/>
- 本实验需要在Linux下完成;
- 注意文件系统数据块和磁盘块的区别, 文件系统数据块大小定为1 KB, 磁盘块大小为512B ;
- 编写报告时需要给出较为详细的用户手册, 命令的使用方式, 支持的功能等。

## ● 作业提交

---

- 本实验要求提交实验报告，最终提交电子档pdf格式，报告命名格式：

**实验报告\_学号\_姓名\_OSLab5.pdf**

- 在作业平台提交zip压缩包，**学号\_姓名\_OSLab5.zip**
- 请务必把握好提交时间，并注意最后一节课程要课堂检查！

# 2023年OS竞赛招新

## 2023年“全国大学生计算机系统能力大赛操作系统设计赛”火热报名中!

- 由教育部高等学校计算机类专业教学指导委员会和系统能力培养研究专家组共同发起，目前已经举办2届，详见<https://hitsz-cslab.gitee.io/os-labs/oscomp/oscomp/>
- 第一届于2021年举办，近200支队伍 参赛。第二届的参赛队伍增加到了 300余支。
- 比赛分为两个赛道：**内核赛道** 和 **功能赛道**

- **写个操作系统而已，有手就行；**
- 对操作系统的完整体系充满兴趣；
- 对内核优化充满想法

• **内核赛道欢迎你!**



Xv6, 不过如此

- 希望尝试 科研 or 提前接触 工业界
- 希望对 前沿、未知 的技术进行探索
- **功能赛道欢迎你!**

第一届:

- “UltraOS”队获得 **内核实现赛道一等奖（赛道排名第一）**；
- “HoitFS”队获得 **功能设计赛道一等奖（赛道排名第一）**；
- “压缩鸭”队获得 **功能设计赛道二等奖**；
- “啊普鲁派哒哒哒; DROP DATABASE teams;”队获得 **内核实现赛道三等奖**。

第二届:

- “FTL OS”队获得 **内核实现赛道一等奖**；
- “Oops”队获得 **内核实现赛道一等奖**；
- “LMY”队获得 **功能设计赛道一等奖**；
- “健康向上好青年”队获得 **内核实现赛道二等奖**；
- “编译通过求求了”队获得 **功能设计赛道二等奖**；
- “F-Tutorials”队获得 **功能设计赛道二等奖**；
- “随便取名不队”获得 **内核实现赛道优胜奖**；
- “追光者队”获得 **功能设计赛道初赛优胜奖**。



群名称: OS比赛-2023招新  
群 号: 250052181

操作系统是计算机领域的一大核心，  
深入理解操作系统，不管以后参与到哪个方向的工作中，都会从中获益。  
欢迎参加操作系统设计赛，体验系统设计的独特魅力。



The background is a light gray with a subtle grid pattern. It features several circles of different sizes and colors (dark blue and white) scattered across the frame. A large white circle with a dark blue border is positioned in the upper left, containing the word 'THANKS' in red. Other circles of various sizes are placed around it, some solid and some with borders.

**THANKS**

同学们，  
请开始实验吧！