華東理工大學

EAST CHINA UNIVERSITY OF SCIENCE AND TECHNOLOGY

《 操作系统课程设计 》

实验报告本

班　级：　计 172

组　号：　5

学号/姓名：　李震　10170811

学号/姓名：　杨博涵　10170810

学号/姓名：　刘浩　10170808

指导教师：　叶琪

信息科学与工程学院

2020 年 6 月

一.课程设计题目

Linux 二级文件系统

二.课程设计内容和要求

李震：底层位操作和工具类的实现；

刘浩：文件增删改查逻辑的实现；

杨博涵：硬盘结构设计和用户模块实现。

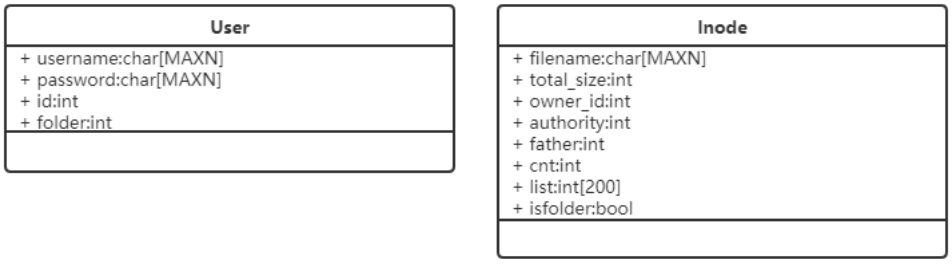
三. 软件系统设计

文件系统业务中，类和实体较少，与用户交互和内部操作的逻辑复杂，所以使用了少量的面向对象，主要采用面向过程的方法进行设计。

关于面向对象设计相关如下：

对象模型：

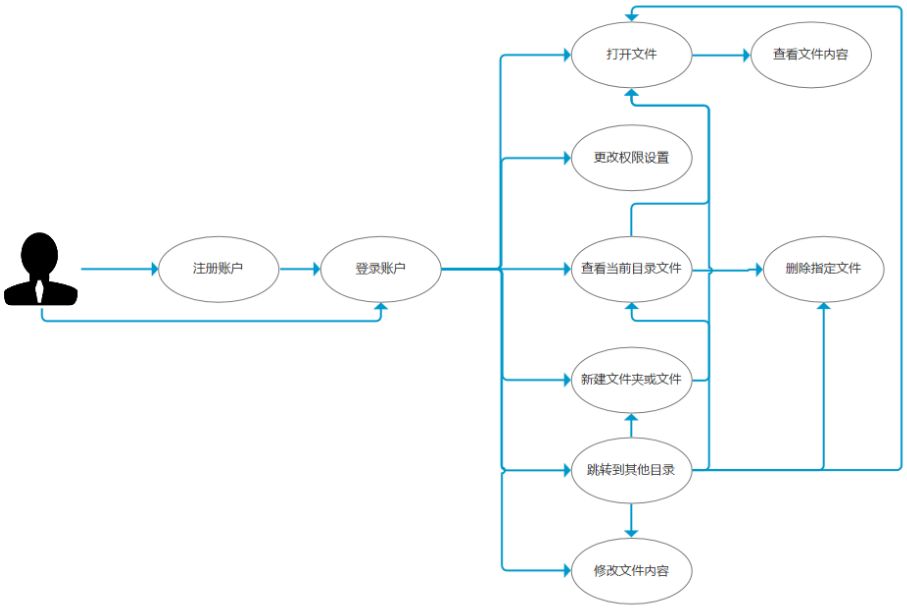
类图：



用户：包括用户名、密码、用户 id 和用户文件夹 id。
Inode：每个文件(夹)对应一个 inode，包括文件名、文件大小、文件所有者 id、保护码（9 位权限码）、父文件夹 id。如果是文件夹，列表存储文件夹下的文件 id；如果是文件，列表存储数据块 id。

功能模型：

使用用例图方式展示



基于此功能模型，在一开始设计好系统实现的相关功能。

函数一览表：

```
void read_from_disk(unsigned char* addr,bool& x)
从磁盘 addr 地址读入一个 bool 存入 x 中。
void write_to_disk(unsigned char* addr,bool x)
将 bool x 写入磁盘的 addr 地址。
void read_from_disk(unsigned char* addr,int& x)
从磁盘 addr 地址读入一个 int 存入 x 中。
void write_to_disk(unsigned char* addr,int x)
将 int x 写入磁盘的 addr 地址。
void read_from_disk(unsigned char* addr,char& x)
从磁盘 addr 地址读入一个 char 存入 x 中。
void write_to_disk(unsigned char* addr,char x)
将 char x 写入磁盘的 addr 地址。
void read_from_disk(unsigned char* addr,struct User& x)
从磁盘 addr 地址读入一个 User 类存入 x 中。
void write_to_disk(unsigned char* addr,struct User& x)
将 User x 写入磁盘的 addr 地址。此处 x 使用引用类型节省内存开销。
void read_from_disk(unsigned char* addr,struct Inoder& x)
从磁盘 addr 地址读入一个 Inode 类存入 x 中。
void write_to_disk(unsigned char* addr,struct Inode& x)
将 Inode x 写入磁盘的 Inode 地址。
int path_analyzer(int id,char* path)
返回从编号为 id 的文件夹经过路径 path 后到达的文件 id，如果路径错误则返回-1。
int fill_data(unsigned char* content)
```

24. 寻找 bitmap 中的空位，将 content 中的数据写入数据块，返回写入的数据块块号。
25. `void erase_data(int p)`
26. 释放块号为 p 的数据块。
27. `unsigned char* get_data(int p,int len)`
28. 从块号为 p 的数据块读取 len 个字节并返回。
29. `int copy_data(int p,int len)`
30. 创建块号为 p 的数据块中前 len 个字节数据的副本，寻找空数据块写入，返回写入的块号。
31. `int create_file(Inode &x,unsigned char* content=NULL)`
32. 创建 Inode x 对应的文件：在 Inode 块中寻找空位存储 x，如果 x 是文件，还要将 content 中的数据写入数据块并将块号存入 x 的 list 中。
33. `bool judge_authority(int id,Inode &x,int authority)`
34. 判断用户 id 是否对 Inode x 对应的文件有 authority 权限。
35. `bool delete_file(int id,int uid=user_id)`
36. 用户 uid 为操作者，删除 id 对应的文件，如果是文件夹则递归删除其中所有文件，如果权限错误返回 **false**。
37. `bool open_file(Inode &x,int uid=user_id)`
38. 用户 uid 为操作者，打开 Inode x 对应的文件，如果权限错误返回 **false**。
39. `bool read_file(Inode &x,int uid=user_id)`
40. 用户 uid 为操作者，读取 Inode x 对应的文件，如果权限错误返回 **false**。
41. `bool write_file(int id,int uid,unsigned char* content)`
42. 用户 uid 为操作者，将 content 中的数据写入 id 对应的文件。
43. `bool judge_include(int from_id,int dst_id)`
44. 复制操作前使用，判断源文件夹 from_id 是否包括目的文件夹 dst_id。
45. `bool copy_all(int from_id,int dst_id)`
46. 将 from_id 对应的文件(夹)复制到 dst_id 目录下，如果权限错误返回 **false**。
47. `void analyze_super_block()`
48. 程序启动初始化时使用，加载超级块。
49. `void initialize()`
50. 程序初始化，加载虚拟硬盘 simdisk 到内存，加载超级块。
51. `void refresh()`
52. 将内存写入虚拟硬盘 simdisk。
53. `void write_test_data()`
54. 初始化空虚拟硬盘，将硬盘分区设计写入超级块，创建根文件夹 home。
55. `User* find_user(char *username)`
56. 返回用户名为 username 的 User 指针。
57. `void create_user(char* username,char* password)`
58. 新建用户，用户名 username，密码 password。
59. `void delete_user(char* username,char* password)`
60. 删除用户，用户名 username，密码 password。
61. `void process_option(int x)`
62. 进行第 x 种操作。
63. `void work()`
64. 监听用户输入，识别相应的操作。
65. `int main()`

66. 主函数。

四.算法设计

inode 中有父目录 id 和各个子目录 id，构成了一个树形结构，根是 home。各种文件操作都需要访问当前节点的父目录和子目录，也就是遍历出边。

pwd 显示当前目录的实现中，从当前目录不断寻找父目录，将经过的目录压栈，最后将栈中元素全部弹出并输出。

bitmap 的操作中，由于无法创建 bit 型变量，只能用一个 unsigned char 存储 8 个 bit，在 8 个 bit 中操作某一个需要进行位运算。

复制、删除目录都需要对该目录下的所有文件进行操作，也就是对子树进行操作，使用了 dfs 进行处理。

五.小结

1.课程设计中遇到的问题和解决方法

在使用内存模拟外存的过程中，发现 C++中的 bool 竟然是一个字节而不是一个 bit，改用位域也经常报错。于是使用 100*1024*1024 长度 unsigned char 数组模拟 100M 的硬盘。这样一来又有了新的问题，对 unsigned char 数组进行二进制操作非常不方便，于是我们先实现了一些将各种类型的数据在 unsigned char 数组中的写入或读取的工具函数。这样一来后面的工作就可以使用这些工具类跳过底层操作的不便了。

有时在 debug 模式和运行模式下程序输出不一样，后发现是有一部分内存释放过早。这个错误和我们使用的 GCC 版本有关。

新建用户的编号怎么决定也是一个问题，我们在用户信息块中放了一个 int 用于记录用户编号的自增标号。

在开发过程中，发现向模拟硬盘 simdisk 中写入数据只能全部覆盖或者向后追加，而不能指定部分覆盖（或许有更底层的函数可以实现）。而如果频繁全部覆盖 100M 会导致速度很慢，因此我们将新改动暂存在内存中，当用户输入 refresh 指令时才将内存写入硬盘。

2.课程设计还存在哪些问题

实际上的文件系统非常复杂，我们三个人仅凭几天时间无法完成，因此我们按照实验指导书的标准进行了非常多的简化，即使这样，我们还是能想到用户的各种操作会导致的问题。例如如何在原先文件内容的基础上修改，我们为了简化逻辑，设计为了文件整体覆盖；设计中文件名、用户名、密码固定最大 31，一个文件的子目录数量最多 200，单个文件最大 200K。

这些都是我们简化文件系统过程中做出的妥协。

在设计过程中也没有考虑 bitmap 查找和文件查找的时间复杂度问题。想要优化可能需要使用空闲链表等数据结构，不过也会使得代码更加难以控制。

加入权限系统，尤其是更改权限 chmod 操作后，对于权限的控制更加复杂，而我们对于 Linux 权限系统了解优先，对于文件权限做了最简单的限制和判断，当更改文件的权限可能会产生很多我们没考虑到的问题。

3.心得体会

李震：

在本次实验中，对 Linux 文件系统有了更深的了解。我们根据自己的理解和简化，基于实验指导书要求的指令集，并且扩充了一些必要的指令，完成了较为基础的的文件系统。在实现过程中遇到了一些底层问题，也有很多设计逻辑问题，我们对这些问题进行了深入的思考，体会到了 Linux 文件系统设计的合理和健全性。所以在实验中，我们按照 Linux 文件系统的结构，设计并实现了我们自己的文件系统，使用了超级块、位示图法以及 inode 等结构，提升了编码和解决 bug 的能力。

我们也认识到了编码前理清设计思路和写好文档的重要性，这些工作会使得后续开发中少走很多弯路。我们这些工作其实并不够，导致最后才意识到一些模块的设计出现了问题，例如用户信息的存储、空闲 inode 的查找以及权限判断代码的统一化。这些问题一定程度上加大了编码的难度，也称为了本项目的遗憾之处。

其次，文件系统的各种 exception 判断非常繁琐，我们也尽量将代码模块化，各个层级判断各自模块中的 exception 并返回。即使这样，exception 判断部分的代码也占据了很多篇幅，对代码阅读造成了很大的影响。而真正 Linux 系统比我们做的 exception 判断复杂的多，其代码复杂性可想而知，只有基于非常完善的开发流程，才能逐步做出如此庞大的系统。有了此项目的经历，我对 Linux 文件系统以及科学设计模式有了更深的理解，以后也会继续实践和完善自己的认识，应用到更多更复杂的项目开发中。

刘浩：

在本次实验中，一开始决定做一个基于 Linux 操作系统的文件管理系统，在一开始基于实验要求以及对于文件系统的理解，以面向对象的类设计方式设计用户，定义一系列函数来完成基本文件系统的相关功能，在这其中，对于文件系统的层次结构有了更加深刻地认识，从组织层次的角度，应用程序，逻辑文件系统.....再到驱动以及设备，这样一个结构每层都利用底层的功能实现更高级的抽象，甚至把文件变成用户便于理解的方式。

对于文件系统增删改查的逻辑代码实现，也让我对于 Linux 在这方面的设计合理性有了充分认识，让自己在实现相关基础功能时能够考虑好一些特殊情况（例如其他用户删除文件

夹时，子目录每个文件对于删改操作的权限问题，如何处理不能删改的目录文件），做好相关处理，让系统功能得以完善。

这次实验过后，让我体会到了实现一个文件系统的复杂性，在对于自己的文件系统做出很多简化之后，才得以完成一个基础的文件系统功能，理解了市面上存在的例如 Linux 以及 windows 系统的逻辑紧密性以及其庞大的功能性。这会使我在日后的工作或者是学习中，更加注重逻辑的紧密，更加会注重，发现一些代码逻辑问题，加强自己在发现以及解决这类问题的能力。同样地，对于一个项目实现最开始的需求分析，功能分析，也会使我在以后的工作中，在开发一个项目之前，认真做好需求分析，为代码的编写打下良好基础的这样一个意识记忆犹新。

杨博涵：

在本次实验中，我们做的是 Linux 二级文件系统，我在其中负责硬盘结构设计和用户模块实现。在设计实现过程中遇到了许多底层和逻辑问题，通过实验指导书和网上搜索最终解决了问题。对硬盘结构有了进一步的理解。在实验中，我们按照 Linux 文件系统的结构，设计并实现了我们自己的文件系统，使用了超级块、位示图法以及 inode 等结构，提升了编码和解决 bug 的能力。

在实验中，我们认识到思路比写代码重要了许多，一个好的程序十分依赖一个好的思路，而我们在初期的思路不是十分清晰也导致了我们遇到了许多困难，最后通过网上相关程序的借鉴和学习，确定硬盘结构选择超级块、用户信息块、位图块、inode 块和数据块的模式。用户模块其实存放用户总数和用户自增标号，。后面线性存储用户名、密码、 和用户文件夹的 inode_id。系统的底层并没有用 bool 数组实现，因为在过程中出现了许多的错误，最终 使用了 数组模拟硬盘。

本次实验中，对 Linux 文件系统有了更深的了解。随着实验的进行，和对文件系统的不断完善，我们对磁盘系统的底层设计有了更清楚的了解，这也是第一次模拟磁盘系统的底层结构，对各种块位置进行准确的操作。我们根据自己的理解和简化，基于实验指导书要求的指令集，并且扩充了一些必要的指令，完成了较为基础的的文件系统。

4. 其它需要补充的问题

代码和文档 github 地址 https://github.com/lichlzh/OS-experiments

六. 参考文献

Linux 文件权限
https://www.cnblogs.com/bbox/p/9974773.html
超级块

https://baike.baidu.com/item/super%20block/6750941?fr=aladdin

ext2

https://baike.baidu.com/item/Ext2/822106?fr=aladdin

课程设计过程记录

日期	时间	实验内容及过程	自我评价
6.29	1-8 节	了解 Linux 二级文件系统，完成文件系统的结构设计和各个模块的内部结构。	第一天虽然对所有的实验都不太了解，但通过多方面查询最终选择了 Linux 二级文件系统，并对结构进行了设计。
6.30	1-8 节	确定并完成底层二进制操作，例如将各种数据写入读出内存虚拟硬盘。	底层操作是整个文件系统除了结构设计外最终要的部分，完成这部分算是最困难的
7.1	1-8 节	实现用户注册、登录、退出和注销操作	在完成底层操作后，用户操作实现起来相对简单了许多，但是还是不太轻松
7.2	1-8 节	实现文件(夹)创建、删除和写入操作	相较于前一天，对各种操作的实现轻松了许多
7.3	1-8 节	研究并加入权限系统，设计权限码内容以及各个操作的权限	相较于前一天，对各种操作的实现轻松了许多
7.4	1-8 节	实现 copy 操作，实现了判断目录之间是否包含以及整个目录内容的复制	相较于前一天，对各种操作的实现轻松了许多
7.5	1-8 节	完善文档，查找并修改文件重名、权限错误等 bug	最后一天，对实验项目进行一些收尾检查工作