操作系统课程设计

姓名：田野

学号：2017329621125

班级：计算机科学与技术（全英文）17（1）

指导老师：郭奕亿

目录

1 引言

随着计算机的普及，家家户户都拥有了计算机，普通用户使用计算机必不可少的部分就是与文件管理系统打交道。用户通过文件管理系统执行创建、打开、复制、移动等操作，并且通过用户权限、用户分组等功能实现文件的权限管理，因此设计一个良好的文件管理系统是至关重要的。

1.1 任务要求

用户在使用操作系统时，必不可少的一项功能就是文件管理系统，就操作系统层面而言，一切皆是文件，所以一个好的文件管理系统对于操作系统而言是必不可少的。操作系统通过文件系统来轻松地存储、定位、提取数据。文件系统有两个不同的设计问题。第一个问题是如何定义文件系统对用户的借口。这个任务设计定义文件及其属性、文件所允许的操作、组织文件的目录结构。第二个问题是创建数据结构和算法来将逻辑文件系统映射到物理外存设备上。

为实现一个多用户系统理解文件系统的层次结构，本系统要提供一下几个功能：

1）基于多级目录实现，文件目录项采用 FCB 或索引节点。

2）提供文件共享及保护手段。

3）实现基本的目录管理及文件管理功能：

（1）目录内容显示

（2）创建/删除目录

（3）更改当前目录

（4）创建/删除文件

（5）读/写文件

（6）打开/关闭文件等。

4）实现用户权限管理、组管理功能。

1.2 选题

本次所选的题目为 B1 和 C1。即用 JAVA 语言模仿“生产者—消费者问题”和设计一个多用户文件系统，理解文件系统的层次结构，完成基本的文件系统 create、open、close、read/write 等基本功能，并实现文件保护操作。

1.3 当前发展概述

当前文件系统是操作系统用来明确存储设备，即是在存储设备上组织文件的方法。操作系统中负责管理和存储文件信息的软件成为文件管理系统，简称为文件系统。文件系统是由三部分组成：文件系统的接口，对于对象操纵和管理的软件集合，对象以及属性。从系统的角度来看，文件系统是对于文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体点说，它主要负责为用户建立文件、存入、读出、修改、转储文件、控制文件的存取，当用户不再使用时撤销文件等功能。

同时，为了防止由于系统崩溃或电源突然中断而导致正在进行的文件操作中断所造成的数据丢失，ext2 文件系统就必须在每个数据块创建或修改后即刻写入磁盘。磁盘的寻道操作对于 cpu 来讲是如此的之长，为了提高性能，所以写操作数据被缓存，写操作被延迟。但这样也带来了数据丢失的风险，假如数据还没来得及写入磁盘,电源突然中断,数据将会丢失。为了解决这个问题，所以产生了 ext3 文件系统。

日志文件系统(ext3)如今已经成为 Linux 的缺省文件系统。日志文件系统就是专门为了那些厌倦了一直盯着启动时 fsck (即文件系统- 致性检查)的人而设

计的(日志文件系统同样适用于希望文件系统具有故障恢复能力的群体)。如果系统采用传统的未提供日志功能的文件系统,那么操作系统在检测到系统为非正常关机时，会使用 fsck 应用程序来执行一致性检验。这个应用程序会扫描文件系统,并且修复任何可以安全修复的问题。而且在一些情况下，当文件系统损坏严重时,操作系统就会启动到单客户模式，由用户进行修复。

网络文件系统，英文名为 NetworkFileSystem(NFS)。它是由 SUN 公司研制出的 UNIX 表示层协议,它可以使使用者访问网络上别处的文件,就像是在使用自己的计算机一样。NFS 是基于 UDP/IP 协议的应用，它的实现主要是采用远程过程调用 RPC 机制，RPC 提供了一组与机器、操作系统以及低层传送协议无关的存取远程文件的操作。而 RPC 则采用了 XDR 的支持。XDR 则是一-种与机器无关的数据描述编码的协议,它以独立与任意机器体系结构的格式对网，上传送的数据进行编码和解码,支持在异构系统之间数据的传送。NFS 的主要特点有:提供透明文件访问以及文件传输、不需要改变现有的工作环境便可以扩充新的资源或软件,高性能，可灵活配置等特点。NFS 允许计算的客户一服务器模型。服务器实施共享文件系统，以及客户端所连接的存储。客户端实施用户接口来共享文件系统,并加载到本地文件空间当中。

1.4 可行性分析

（1）技术条件可行性分析

本文件管理系统是一个操作系统层面的应用程序，现有的开发技术已经非常成熟，估计利用现有技术完全可以达到功能目标，考虑到开发期限较为充裕，预计可以在规定期限内完成开发。

（2）经济可行性分析

目前，计算机的价格已经十分低廉，性能却有了长足的进步，而本系统的开发,完善了操作系统的功能,为此主要表现为:本系统的运行可以节省许多资源、本系统的运行可以大大提高使用人员的工作效率、本系统可以是敏感文档更加安

全，可靠，所以本系统在经济上是可行的。

因此，经上述可行性分析，本系统的研制和开发完全达到预期目标。

1.5 选题意义

由于内存通常太小，并不足以永久保存所有数据和程序，所以计算机系统必须提供外存以备份内存。现代计算机系统采用磁盘作为主要在线存储以保存信息。文件系统由两个不同部分组成，一组文件和目录结构。文件系统位于设备上。文件是一组有创建者所定义的相关信息的集合。文件系统为存储与访问磁盘上的数据与程序提供机制。操作系统将文件映射到物理设备上。文件通常按目录来组织，以便于使用。对于绝大多数用户而言，文件系统是操作系统中最为可见的部分。它提供了在线存储和访问计算机操作系统和所有用户的程序与数据的机制。

因此为操作系统提供一个设计良好，用户友好的文件系统对改善用户体验、提升系统性能的功能。

2. 需求分析与设计

2.1 需求分析

2.1.1 功能性需求

要求实现一个多用户系统，并且本系统要提供一下几个功能：

1）基于多级目录实现，文件目录项采用 FCB 或索引节点。

2）提供文件共享及保护手段。

3）实现基本的目录管理及文件管理功能：

（1）目录内容显示

（2）创建/删除目录

（3）更改当前目录

（4）创建/删除文件

（5）读/写文件

（6）打开/关闭文件等。

4）实现用户权限管理、组管理功能。

2.1.2 非功能性需求

要求所设计的多用户文件管理系统具有良好的用户交互逻辑，用户操作需要简单方便。对于不熟悉改系统的用户提供引导，帮助其在短时间内熟悉系统的功能。

要求对文件管理系统的设计考虑到各种意外情况，并能从各种意外情况中恢复，接着正常运行。

2.1.3 设计约束

要求该系统至少能运行在 Linux 环境，开发时应该使用 C++等与操作系统进

行交互的开发语言。

2.2 系统框架和流程

下面是多用户文件系统的用例图，图中展示了普通用户与管理员拥有功能的不同。

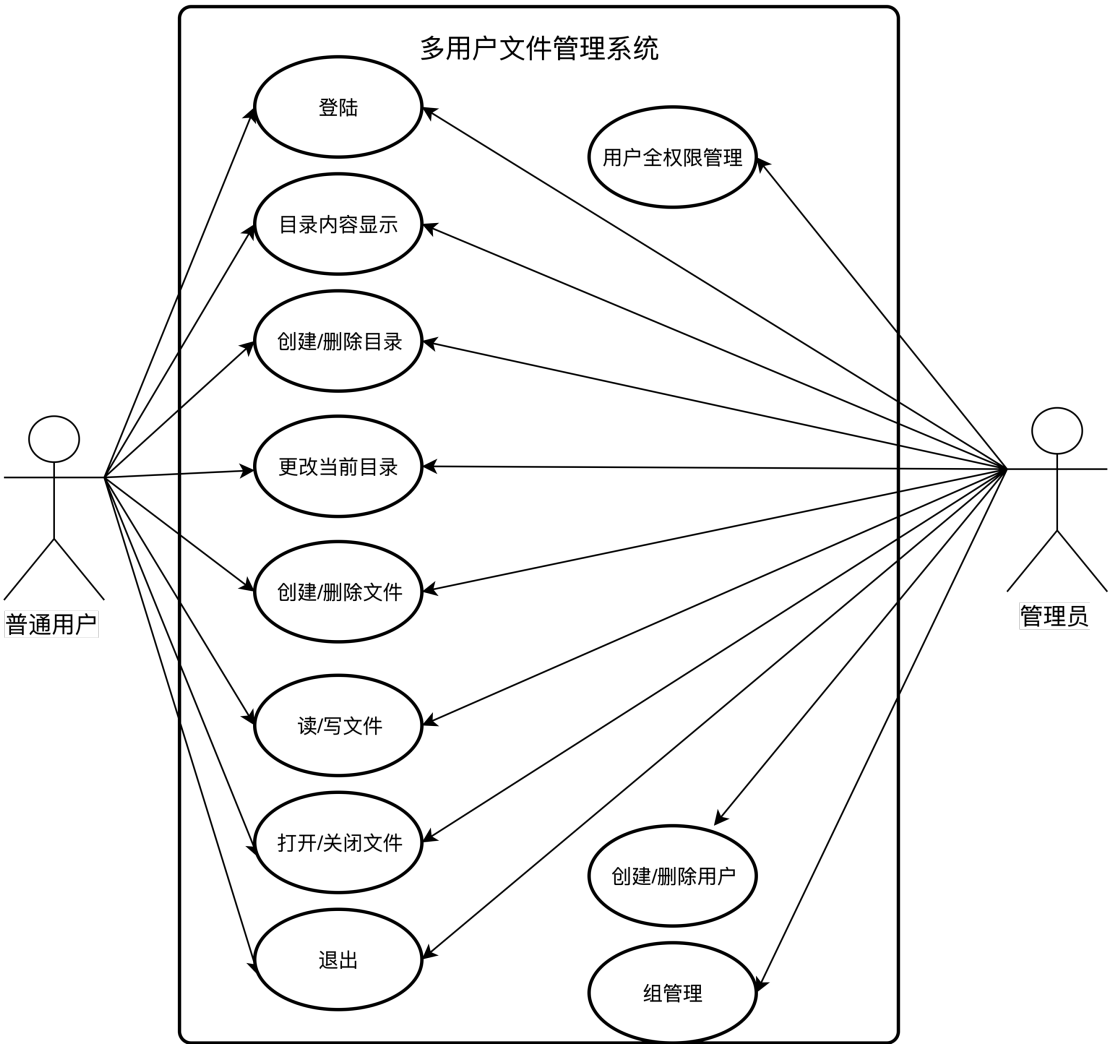

图 2.1 文件管理系统用例图

下图是系统流程图，用户首先需要登陆，根据角色的不同有着不同的操作权限。普通用户有目录内容都显示、创建/删除目录、更改当前目录、创建/删除目录、读/写文件、打开/关闭文件功能，管理员在这些功能至上还有组管理、创建/删除用户、用户权限管理这些功能，如图 2.2 所示

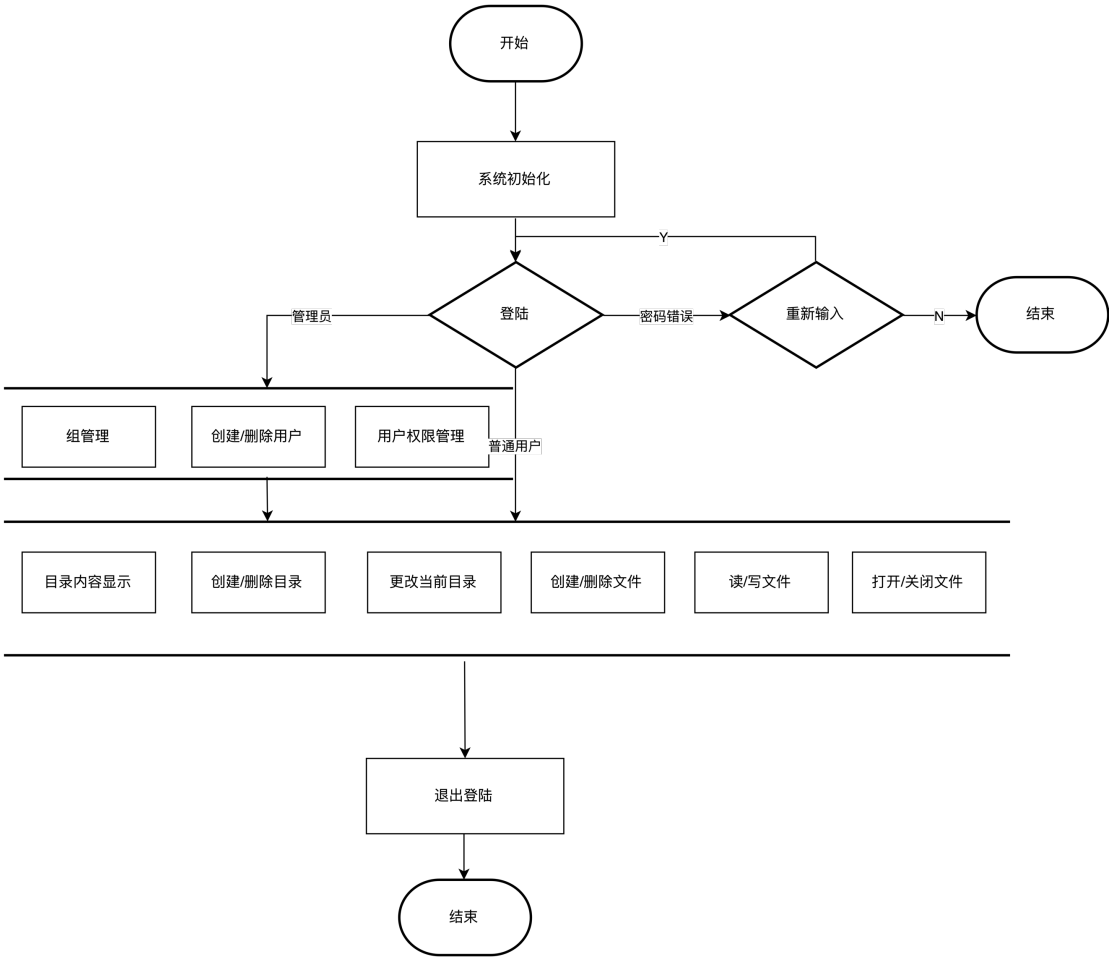

图 2.2 系统流程图

下图是系统结构图，系统总共分为用户管理模块、文件夹管理模块、文件管理模块、控制输出模块这四大模块，每个模块有着不同的功能。通过模块间功能的配合，完成了系统的所有功能，系统结果如图 2.3 所示

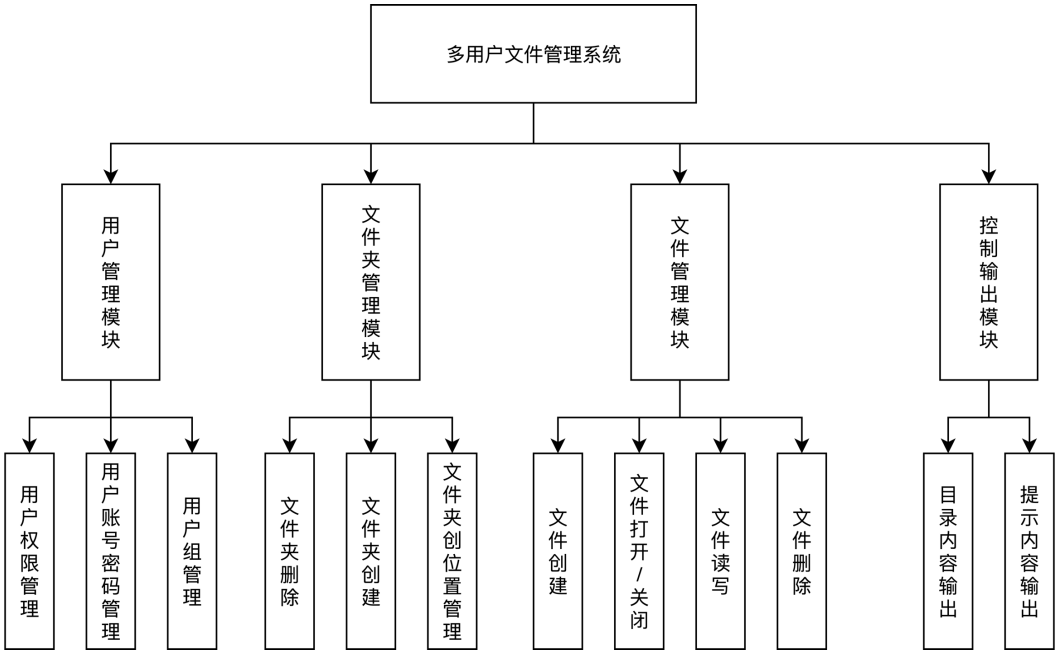

图 2.3 系统结构图

3. 数据结构

常量定义：

```
//常量定义


// NADDR 为文件可使用的最大数量节点
const unsigned int NADDR = 6;


// BLOCK_SIZE 为一个 block 的大小
const unsigned short BLOCK_SIZE = 512;


//文件最大尺寸
const unsigned int FILE_SIZE_MAX = (NADDR - 2) * BLOCK_SIZE + BLOCK_SIZE /
sizeof(int) * BLOCK_SIZE;


//block 的数量
const unsigned short BLOCK_NUM = 512;


//inode 索引节点大小
const unsigned short INODE_SIZE = 128;


//inode 数量
const unsigned short INODE_NUM = 256;


//inode 的起始位置
const unsigned int INODE_START = 3 * BLOCK_SIZE;


//数据起始位置
const unsigned int DATA_START = INODE_START + INODE_NUM * INODE_SIZE;
```

```
//文件系统支持的用户数量
const unsigned int ACCOUNT_NUM = 10;

//子目录的最大数量
const unsigned int DIRECTORY_NUM = 16;

//文件名的最大长度
const unsigned short FILE_NAME_LENGTH = 14;

//用户名的最大长度
const unsigned short USER_NAME_LENGTH = 14;

//用户密码的最大长度
const unsigned short USER_PASSWORD_LENGTH = 14;
```

inode 设计

```
struct inode
{
    unsigned int i_ino;          //inode 号.
    unsigned int di_addr[NADDR];   // 存储文件的数据块数。
    unsigned short di_number;    // 关联文件数。
    unsigned short di_mode;       //文件类型.
    unsigned short icount;       //连接数
    unsigned short permission;    //文件权限
    unsigned short di_uid;       //文件所属用户 id.
    unsigned short di_grp;       //文件所属组
    unsigned short di_size;       //文件大小.
    char time[83];
};
```

超级块设计：

```
//超级块设计
```

```
struct filsys
{
    unsigned short s_num_inode;          //inode 总数
    unsigned short s_num_finode;         //空闲 inode 数.
    unsigned short s_size_inode;         //inode 大小.
    unsigned short s_num_block;          //block 的数量.
    unsigned short s_num_fblock;         //空闲块的数量.
    unsigned short s_size_block;         //block 的大小.
    unsigned int special_stack[50];
    int special_free;
};
```

目录设计：

```
//目录设计
struct directory
{
    char fileName[20][FILE_NAME_LENGTH];    //目录名称
    unsigned int inodeID[DIRECTORY_NUM];    //inode 号
};
```

账户设计：

```
//账户设计
struct userPsw
{
    unsigned short userID[ACCOUNT_NUM];              //用户 id
    char userName[ACCOUNT_NUM][USER_NAME_LENGTH];      //用户名
    char password[ACCOUNT_NUM][USER_PASSWORD_LENGTH];   //用户密码
    unsigned short groupID[ACCOUNT_NUM];             //用户所在组 id
};
```

全局变量：

```
//全局变量
```

```
FILE* fd = NULL; //文件系统位置


//超级块
filsys superBlock;


//1 代表已经使用，0 表示空闲
unsigned short inode_bitmap[INODE_NUM];


//用户
userPsw users;


//用户 id
unsigned short userID = ACCOUNT_NUM;


//用户名
char userName[USER_NAME_LENGTH + 6];


//当前目录
directory currentDirectory;
```

4. 关键技术

4.1 成组链接法

在 UNIX 系统中，将空闲块分成若干组，每 100 个空闲块为一组。其中仅有一组会被调入内存，被调入内存的块称为超级块。每组的第一空闲块登记了下一组空闲块的物理盘块号和空闲块总数。如果一个组的第二个空闲块号等于 0，则有特殊的含义，意味着该组是最后一组，即无下一个空闲块。成组链接法的基本架构如图 4.1 所示（为简化说明未完全画完盘块）：

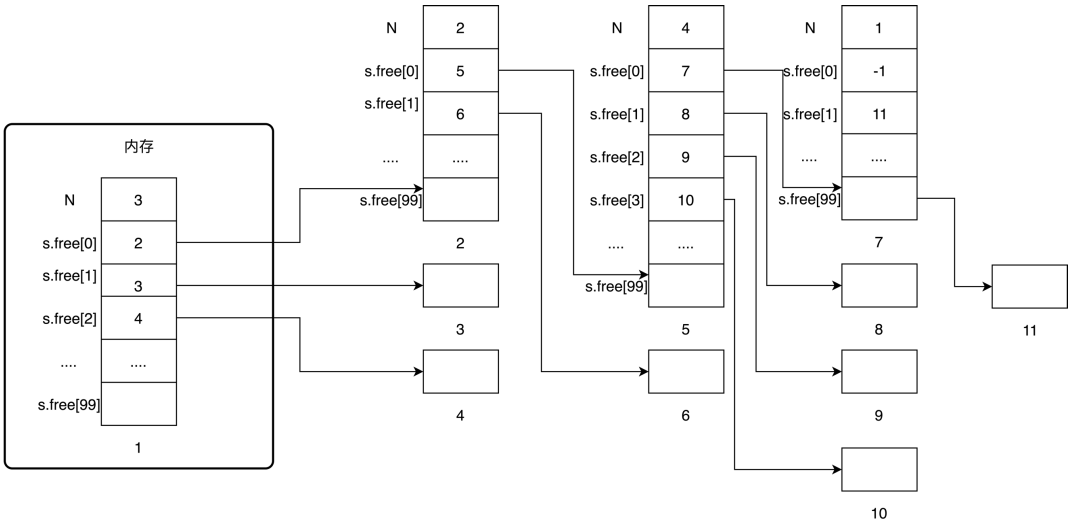

图 4.1 成组链接法结构

在需要空间分配时，假如要为一个文件分配一个盘块，系统将会从超级块中出栈一个盘块，然后把这个编号对应的盘块分配给文件。以图 4.1 为例，此时若需要分配一个盘块，则 4 号盘块分配给文件，4 号出栈，并且超级块中的 N 就会减 1 变成 2，如果还不够继续分配，那么就继续出栈，下一个出栈的编号是 3，同样的 N 接着减 1 变成 1，也就是说此时超级块只剩下一个盘块，这个盘块含有一个栈，它保存了后续盘块的信息。当还需要分配盘块时，需要将它里面的栈和 N 保存到超级块中。假如这时仍需要给文件分配盘块，那么就将 2 的信息先保存到超级块中，然后把 2 分配给文件，然后重新更新一下指针。超级块中的内容被原来盘块 2 中的内容所覆盖，也就是原来盘块 2 中保存的后续节点的信息被转

移到了超级块中，结果如图 4.2 所示。

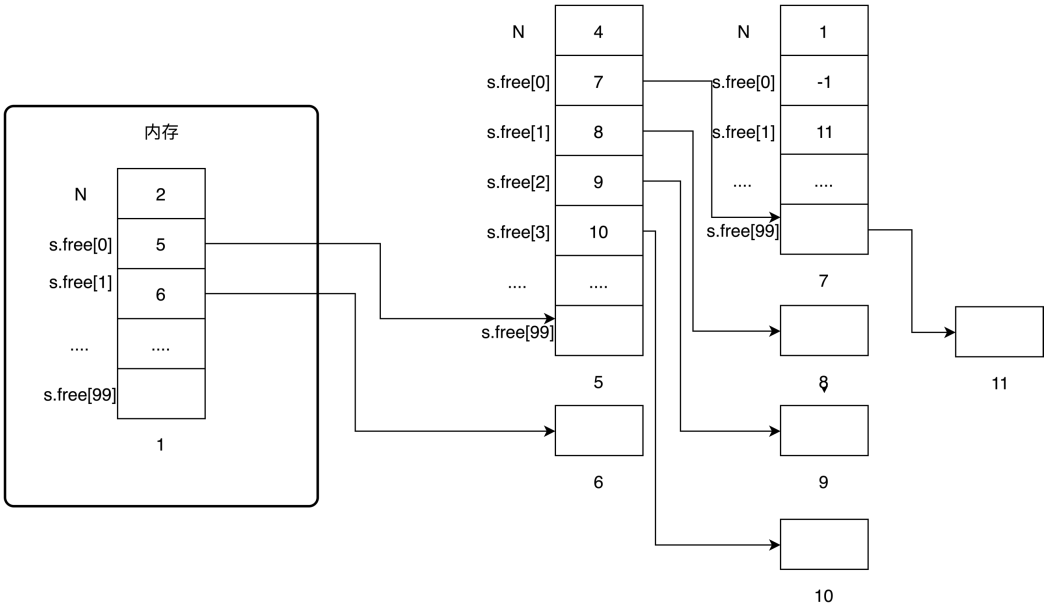

图 4.2 成组链接法查找空闲块

在需要空间回收的时候，直接让其入栈超级块，并更新指针。假如此时来了一个磁盘块 4，则直接将盘块 4 压入栈中，之后更新 N。就这样来一个盘块就压入一个。当栈满了之后（为了方便演示，假设下一步就满了），就会把超级块的内容复制到新回收的盘块中，假如新回收的盘块是 3，就把当前超级块中的内容复制到盘块 3 中，然后更新一下指针，盘块 3 与之后的盘块建立了联系，然后根据盘块 3 更新超级块的栈和 N，显然 N 为 1，因为超级块所指的下一组只有这个盘块 3，并且把盘块 3 压入栈中。这就是成组链接法的空间回收，如图 4.3 所示:

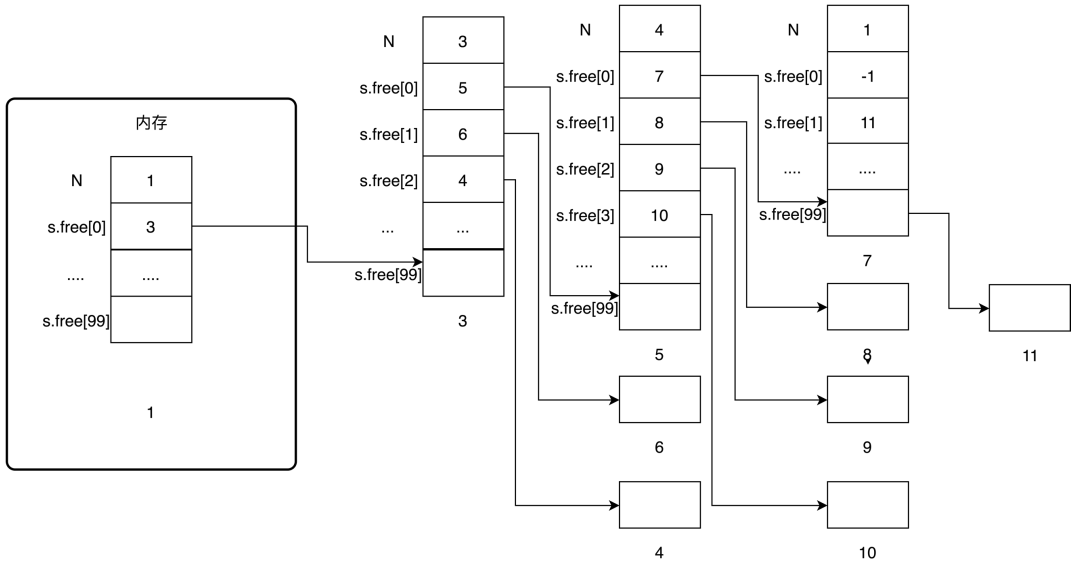

图 4.3 成组链接法空间回收

5. 运行结果

5.1 运行环境

CPU: 2.5 GHz/2.7 GHz

内存：2G 内存

操作系统：Ubuntu 18.04

5.2 运行结果

5.2.1 登陆界面

在使用系统之前首先需要登陆到系统，当用户输入的账号或密码出错时，会提醒用户输入错误，要求重新输入。如图 5.1 所示

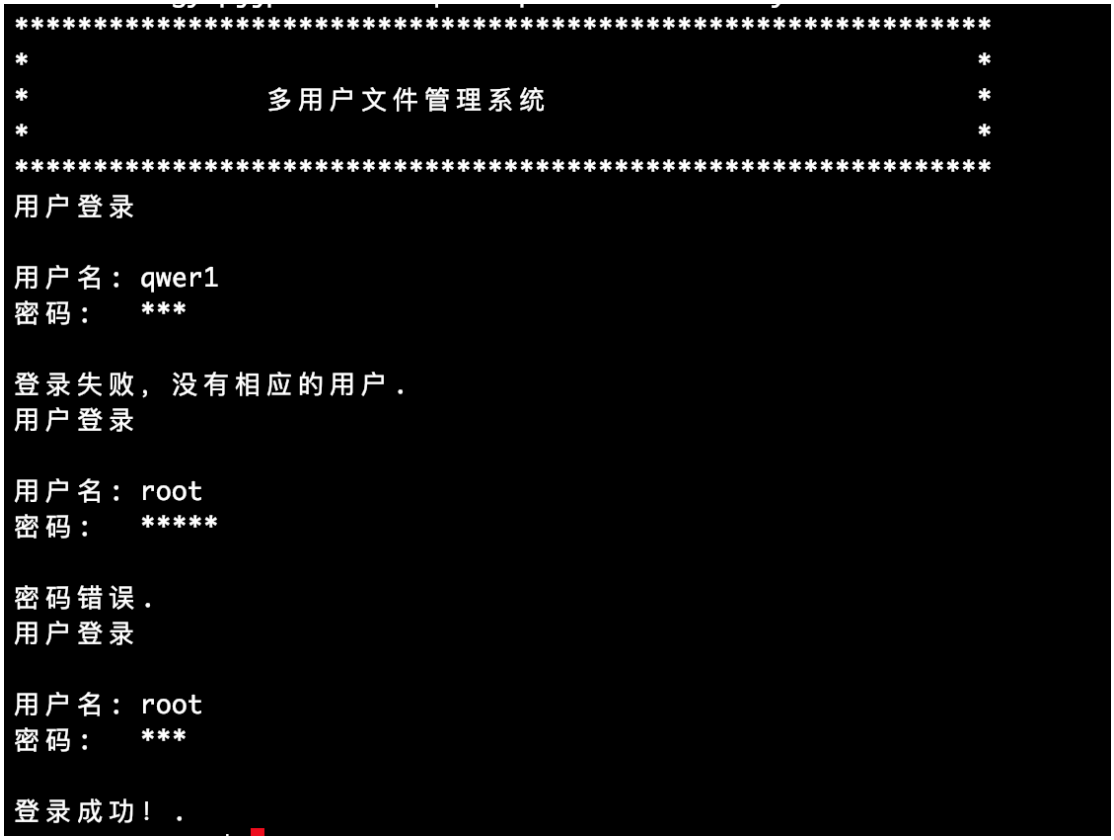

图 5.1 登陆界面

5.2.2 help 指令

在系统中输入 help 指令时，会显示系统支持的所有指令名称及其对应的功能，当需要查找对应指令使用方法时也可使用 help 指令，效果如图 5.2 所示:

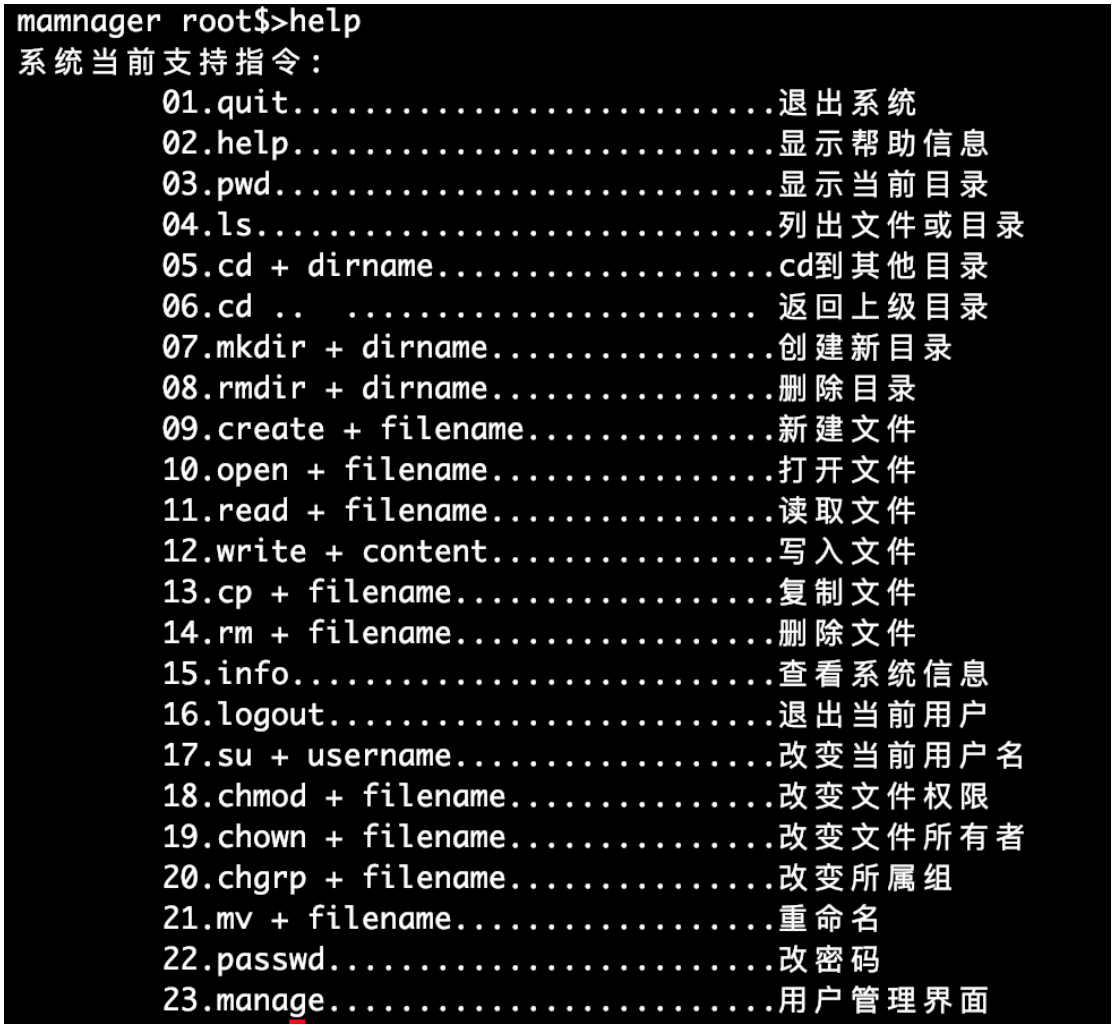

图 5.2 help 指令界面

5.2.3 pwd 指令

在系统中输入 pwd 指令时，系统会显示当前的绝对路径，如图 5.3 所示

图 5.3 pwd 指令界面

5.2.4 ls 指令

在系统中输入 ls 指令时，系统会显示当前文件夹中所有文件的详细信息，如图 5.4 所示

```
mamnager root$>ls

     name       user    group   inodeID Icount  size    permission      time
        .       root    1       0       0       0       drwx------      2019/12/31 00:50:04
       ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
   system       root    1       1       0       320     -r-x------      2019/12/31 00:50:04
```

图 5.4 ls 指令界面

5.2.5 mkdir 指令

当在系统中输入 mkdir + dirname 指令时，系统会在当前目录创建一个命名为 dirname 的次级目录，如图 5.5 所示

```
mamnager root$>mkdir test
mamnager root$>ls

     name       user    group   inodeID Icount  size    permission      time
        .       root    1       0       0       0       drwx------      2019/12/31 00:50:04
       ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
   system       root    1       1       0       320     -r-x------      2019/12/31 00:50:04
     test       root    1       2       0       344     drwxrwxrwx      2019/12/31 01:05:06


mamnager root$>cd test
mamnager root$>pwd
root/test/
```

图 5.5 mkdir 指令界面

5.2.6 create 指令

当在系统中输入 create+filename 指令时，系统会在当前目录创建一个命名为 filename 的文件，如图 5.6 所示

```
mamnager root$>create myfile
mamnager root$>ls

     name       user    group   inodeID Icount  size    permission      time
        .       root    1       2       0       344     drwxrwxrwx      2019/12/31 01:07:38
       ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
   myfile       root    1       3       0       0       -rwxrwxrwx      2019/12/31 01:07:46
```

图 5.6 create 指令界面

5.2.7 open, write, read 指令

在系统中输入 open+filename 指令时，系统会打开命名为 filename 的文件，此时可以输入 write+content 指令进行写入文件操作，之后可以输入 read+filename 指令读出文件内容，如图 5.7 所示

```
mamnager root$>open myfile
mamnager root$>write testcontent
mamnager root$>read

1       testcontent
```

图 5.7 open, write, read 指令界面

5.2.8 chmod 指令

在系统中输入 chmod+filename or dirname 指令，可以为系统文件或文件夹设定访问权限，如图 5.8 所示

```
mamnager root$>chmod myfile
0=文件, 1=目录, 请输入:0
请输入 0&1 串给予权限
格式: rwerwerwe
111111010
mamnager root$>ls

    name       user    group   inodeID Icount  size    permission      time
       .       root    1       2       0       344     drwxrwxrwx      2019/12/31 01:25:34
      ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
  myfile       root    1       11      0       0       -rwxrwx-w-      2019/12/31 01:25:41
```

图 5.8 open, write, read 指令界面

5.2.9 chgrp 指令

在系统中输入 chgrp+filename or dirname 指令，可以修改文件或文件夹所属组，如图 5.9 所示

```
mamnager root$>chgrp myfile
0=文件, 1=目录, 请选择:0
请输入组号:1
mamnager root$>ls

    name       user    group   inodeID Icount  size    permission      time
       .       root    1       2       0       344     drwxrwxrwx      2019/12/31 01:25:34
      ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
  myfile       root    1       11      0       0       -rwxrwx-w-      2019/12/31 01:25:41
```

图 5.9 chgrp 指令界面

5.2.10 chown 指令

在系统中输入 chown+filename or dirname 指令，可以修改文件或者文件夹所属用户，如图 5.10 所示

```
mamnager root$>chown myfile
0=文件，1=目录，请选择:0
请输入用户名:tianye
mamnager root$>ls

    name       user    group   inodeID Icount  size    permission      time
       .       root    1       2       0       344     drwxrwxrwx      2019/12/31 01:25:34
      ..       root    1       0       0       0       drwx------      2019/12/31 00:50:04
  myfile       tianye  1       11      0       0       -rwxrwx-w-      2019/12/31 01:25:41
```

图 5.10 chown 指令界面

5.2.11 manage 指令

管理员在系统中输入 manage 指令，可以进入用户管理界面，在此界面可以查看用户信息，创建以及删除用户，如图 5.11 所示

```
mamnager root$>manage
欢迎来到用户管理！

1.查看 2.创建 3.删除 0.保存 & 退出
1
0       root    1
1       tianye  2

1.查看 2.创建 3.删除 0.保存 & 退出
2
请输入用户名:test
请输入密码:123
请输入 group ID:2
成功！

1.查看 2.创建 3.删除 0.保存 & 退出
0
mamnager root$>
```

图 5.11 manage 指令界面

6. 调试和改进

在编写本系统的时候，其中最困难的部分就是成组链接算法的实现，因为要实现这一算法，会涉及到很多指针操作，只是在平时使用诸如 java, python 之类的高级语言时不会面临的情况，所以在调试指针时花费了大量的时间。

一开始在编写这个算法的时候纯粹就是想到什么就写了什么，所以导致在后面整个代码显得特别杂乱，当需要改动一个小地方的时候，可能关联到这里一点，那里一点，修改起来十分麻烦，甚至可能后面忘记了之间的关联性，要调适很久才能找到修改的地方。后来索性重新写了这部分代码，这次注重的功能之间的分离，让每个模块只负责专门的功能，尽量少在某个模块内想其他的模块传递参数。减少模块之间的耦合性，增加模块的内聚性。

7. 心得和结论

7.1 结论和体会

本系统完成了一个多用户的文件管理系统，具有基本的目录内容显示、创建/删除目录、更改当前目录、创建/删除文件、读/写文件、打开/关闭文件、及用户权限管理、组管理功能。但目前系统还不够完善，鲁棒性仍然很弱，在输入非指定命令格式时可能会出现意想不到的错误。

经过这次的设计，让我明白了在写代码之前动手分析、设计结构等步骤等重要性，不能低估这些没有写代码的时间，相反充分利用好这些设计的时间，设计出一个高内聚低耦合的系统，在编写代码、后期调试时会达到事半功倍的效果。

7.2 进一步改进方向

相较于常用的文件管理系统而言，目前设计出的文件管理系统仍然具有很多方面的不足，例如：在输入时不允许按下删除键，不支持指令可选功能，支持的指令还太少等，如果能再有一些时间，那么接下来的改进方向就是增加系统的鲁棒性，让系统在输入非指定命令时仍能正常工作，增加系统支持的指令，扩展系统的功能，让其达到做到一个真正的文件管理系统能做的所有事情。

主要参考文献

[1] Operating_Systems_Concepts[7th] Abraham Silbershatz