**《操作系统课程设计》**

**实验报告**

徽标, 公司名称

描述已自动生成

**题目: 文件系统实验**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212171**

**姓 名 杨晨**

**2024年 4月**

## 目 录

第一章 实验概述 1

1.1 实验目的 1

1.2 实验内容 1

1.2.1 XXX 1

1.2.2 XXXX 1

1.3 XXXX 1

第二章 实验步骤 2

2.1 XXXX 2

2.1.1 XXXX 2

2.1.2 XXX 2

2.2 XXXX 2

2.2.1 XXX 2

2.2.2 XXX 2

2.3 XXXXX 10

2.3.1 XXXX 10

2.4 XXXXX 12

2.4.1 XXXX 12

2.4.2 XXXX 16

第三章 实验结果分析 22

3.1 XXX 22

3.1.1 XXXX 22

3.1.2 XXXX 22

3.2 XXXXX 22

3.2.1 XXX 22

3.2.2 XXX 24

3.3 XXXX 24

3.3.1 XXXX 24

3.3.2 XXX 26

3.4 XXX 26

3.4.1 XXXXX 26

3.4.2 XXXX 29

3.4.3 XXXX 29

第四章 总结 36

4.1XXXXX 36

# **第一章 实验概述**

##### 1.1 实验目的

(1) 掌握文件系统的工作机理。

(2) 理解文件系统的主要数据结构。

(3) 学习较为复杂的Linux下的编程。

(4) 了解EXT2文件系统的结构。

Linux使用虚拟文件系统(VFS)技术,从而可以支持多达几十种不同类型的文件系统，而EXT2是Linux自己的文件系统。无论是VFS还是EXT2都是采用UNIX的文件系统的模型来构造的。在本实验中,设计了一个二级文件系统，它保留了VFS和EXT2模型的主要结构和功能，是一个比真正的文件系统简单得多，而又能基本体现文件系统理论的程序。最终通过该实验达到理解一般文件系统的原理和实现机制的目的。

通过本实验，实验者能够了解EXT2文件系统的内核基本数据结构各字段的信息。这些信息包括超级块、块组描述符、索引节点等

##### 1.2 实验内容

# **1.2.1 设计并实现一个二级文件系统程序**

(1) 提供以下操作：

* 文件创建/删除接口命令create/delete
* 目录创建/删除接口命令mkdir/rmdir。
* 显示目录内容命令1s。

(2) 创建的文件不要求格式和内容

(3) 具备提供用户登录的功能。

(4) 文件、目录要有权限。

# **1.2.2 EXT2 文件系统结构的分析与跟踪**

(1) 显示内核超级块数据结构的字段信息。

(2) 显示块组数据结构字段的信息。

(3) 显示目录下所有文件的目录数据结构字段信息

(4) 根据输入的文件的inode号显示其inode 数据结构字段信息。

# **第二章 实验步骤**

##### 2.1 二级文件系统的设计与实现

# **2.1.1 程序设计思想**

要求设计一个简单的文件系统,它包含了格式化、显示文件(目录)、创建文件、登录等几个简单命令的实现，而且能完成超级块的读写、inode的读写等过程，是一个比真正的文件系统简单得多，而又能基本体现文件系统理论的程序。在超级块的使用上，采用了操作系统关于这方面的经典理论；而在节点的使用上，主要是模仿Linux的EXT2文件系统。

这是一个面向过程的程序，它的流程图如图

图示

描述已自动生成所示。

# **2.1.2 main.c参考代码**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#include <string.h>

#include "structure.h"

#include "creat.h"

#include "access.h"

#include "ballfre.h"

#include "close.h"

#include "delete.h"

#include "dir.h"

#include "format.h"

#include "halt.h"

#include "iallfre.h"

#include "install.h"

#include "log.h"

#include "name.h"

#include "open.h"

#include "rdwt.h"

#include "igetput.h"

struct hinode hinode[NHINO];

struct dir dir;

struct file sys\_ofile[SYSOPENFILE];

struct filsys filsys;

struct pwd pwd[PWDNUM];

struct user user[USERNUM];

FILE \*fd;

struct inode \*cur\_path\_inode;

int user\_id;

/\*kkkkk\*/

unsigned short usr\_id;

char usr\_p[12];

char sel;

char temp\_dir[12];

main()

{

unsigned short ab\_fd1, ab\_fd2, ab\_fd3, ab\_fd4, i, j;

char \*buf;

int done = 1;

printf("\nDo you want to format the disk(y or n)?\n");

if (getchar() == 'y')

{

printf("\nFormat will erase all context on the disk \n");

printf("Formating...\n");

format();

printf("\nNow will install the fillsystem,please wait...\n");

install();

printf("\n----Login----\nPlease input your userid:");

scanf("%u", &usr\_id);

printf("\nPlease input your password:");

scanf("%s", &usr\_p);

/\* printf("\nsuccess\n");\*/

if (!login(usr\_id, usr\_p))

return;

while (done)

{

printf("\n Please Select Your Operating\n");

printf(" -1----ls\n -2----mkdir\n -3----change dir\n -4----create file\n -0----Logout\n"); /\* 注意\*/

sel = getchar();

switch (sel)

{

case '1': /\*显示目录内容\*/

\_ls();

break;

case '2': /\*创建目录\*/

printf("please input dir name:");

scanf("%s", temp\_dir);

mkdir(temp\_dir);

break;

case '3': /\*改变当前目录\*/

printf("please input dir name:");

scanf("%s", temp\_dir);

chdir(temp\_dir);

break;

case '4': /\*创建文件\*/

printf("please input file name:");

scanf("%s", temp\_dir);

ab\_fd1 = creat(2118, temp\_dir, 01777);

buf = (char \*)malloc(BLOCKSIZ \* 6 + 5);

write(ab\_fd1, buf, BLOCKSIZ \* 6 + 5);

close(0, ab\_fd1);

free(buf);

break;

case '0': /\*退出文件系统\*/

logout(usr\_id);

halt();

done = 0;

default:

printf("error!\nNo such command,please try again.\nOr you can ask your teacher for help.\n");

break;

}

}

}

else

printf("User canseled\nGood Bye\n");

}

# **2.1.3 几个重要的算法处理**

***1)数据块的安排***

初始化数据块位于format.h中，下面是相关的算法描述。初始化数据块的算法流程图如图所示。

NICFREE：常量，为每个块组的大小，程序中设为50。

FILEBLK：常量，为系统允许最多块数，程序中设为512。下面为参考程序源代码：

for (i = NICFREE + 2; i < FILEBLK; i += 50) // 为何要加2，参看149行的注释

{

for (j = 0; j < NICFREE; j++)

{

block\_buf[NICFREE - 1 - j] = i - j;

}

fseek(fd, DATASTART + BLOCKSIZ \* (i - 49), SEEK\_SET);

fwrite(block\_buf, 1, BLOCKSIZ, fd);

} // 当i=502之后，完成文件块502～453的写入；

// 之后文件块512～503不能进行，需要特殊处理

for (i = 503; i < 512; i++)

block\_buf[i - 503] = i;

fseek(fd, DATASTART + BLOCKSIZ \* 503, SEEK\_SET);

fwrite(block\_buf, 1, BLOCKSIZ, fd); // 至此，才完成512块文件块的定位

for (i = 0; i < NICFREE; i++)

{

filsys.s\_free[i] = i + 3; // 从DATASTART的第一个BLOCK作为MAIN DIRECTORY

// 第二个BLOCK作为etc目录

// 第三个BLOCK作为password文件

// 故此i要加3

}

图示

描述已自动生成

所有的512个数据块被分成若干个块组，每个块组拥有50个数据块。每个块组的第一个数据块存放有该块组其他数据块的偏移量，这里使用偏移量来模拟数据在磁盘上的地址。

当前使用块组的各个数据块地址存放在全局变量blockbuf[]当中。

***2)数据块的分配和回收***

数据块的分配和回收由位于ballfre.h的代码完成。

1. balloc()函数用于分配数据块，数据块的分配算法流程图如图所示。下面是部分的参考程序源代码：

图示

描述已自动生成

if (filsys.s\_nfree == 0)

{

printf("\nDisk Full!!\n");

return DISKFULL;

}

i = filsys.s\_pfree;

flag = (i == 0);

if (flag) // 该BLOCK组全部用了

{

fseek(fd, DATASTART + BLOCKSIZ \* (filsys.s\_free[NICFREE - 1] + 1), SEEK\_SET);

// filsys.s\_free[NICFREE-1]+1指向下一个block组的地址块

fread(block\_buf, 1, BLOCKSIZ, fd);

for (i = 0; i < NICFREE; i++)

{

filsys.s\_free[i] = block\_buf[i];

} // 将待用block组的地址读入超级块

filsys.s\_pfree = NICFREE - 1;

free\_block = filsys.s\_free[filsys.s\_pfree];

}

else

{

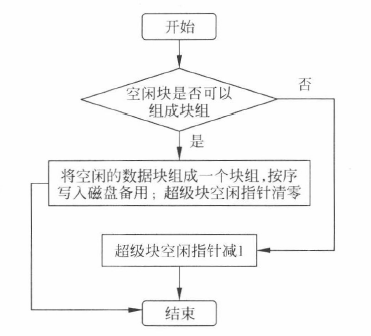
free\_block = filsys.s\_free[filsys.s\_pfree];

filsys.s\_pfree--;

}

其中filsys.s\_nfree便是超级块中指向下一个空闲数据块的指针

1. bfree()函数用于回收数据块，数据块的回收的算法流程图如图所示。下面是部分参考程序源代码：



if (filsys.s\_pfree == NICFREE - 1)

// 表示回收的block已经可以组成一个block组了

{

for (i = 0; i < NICFREE; i++)

{

block\_buf[i] = filsys.s\_free[NICFREE - 1 - i];

}

filsys.s\_pfree = 0;

fseek(fd, DATASTART + BLOCKSIZ \* (filsys.s\_free[0]), SEEK\_SET);

// filsys.s\_free[0]为当前BLOCK组的地址块

fwrite(block\_buf, 1, BLOCKSIZ, fd);

}

else

filsys.s\_pfree++;

1. access()函数用于访问控制，访问控制的算法流程图如图所示。下面是部分参考程序源代码：

switch (mode)

{

case READ:

if (inode->di\_mode & ODIREAD)

return 1;

if ((inode->di\_mode & GDIREAD) &&

(user[user\_id].u\_gid == inode->di\_gid))

return 1;

if ((inode->di\_mode & UDIREAD) &&

(user[user\_id].u\_uid == inode->di\_uid))

return 1;

return 0;

case WRITE:

if (inode->di\_mode & ODIWRITE)

return 1;

if ((inode->di\_mode & GDIWRITE) &&

(user[user\_id].u\_gid == inode->di\_gid))

return 1;

if ((inode->di\_mode & UDIWRITE) &&

(user[user\_id].u\_uid == inode->di\_uid))

return 1;

return 0;

case EXICUTE:

if (inode->di\_mode & ODIEXICUTE)

return 1;

if ((inode->di\_mode & GDIEXICUTE) &&

(user[user\_id].u\_gid == inode->di\_gid))

return 1;

if ((inode->di\_mode & UDIEXICUTE) &&

(user[user\_id].u\_uid == inode->di\_uid))

return 1;

return 0;

default:

return 1;

}

图示

描述已自动生成

***3)文件的相关操作***

1. create()函数用于文件创建，创建文件的算法流程图如图所示。下面是部分参考程序源代码：

图示

描述已自动生成

while (i < USERNUM) // user[]的值由函数login()注册，参看文件log.h

{

if (user[i].u\_uid == uid)

{

user\_id = i;

break;

}

i++;

}

if (i == USERNUM)

{

printf("The user id is wrong.\n");

exit(1);

}

di\_ino = namei(filename);

if (di\_ino != -1) // 文件已经存在

{

inode = iget(di\_ino);

if (access(user\_id, inode, mode) == 0)

{

iput(inode);

printf("\nThe creating access not allowed\n");

return 0;

}

for (i = 0; i < inode->di\_size / BLOCKSIZ + 1; i++)

{

bfree(inode->di\_addr[i]);

}

for (i = 0; i < SYSOPENFILE; i++)

if (sys\_ofile[i].f\_inode == inode)

{

sys\_ofile[i].f\_off = 0;

}

for (i = 0; i < NOFILE; i++)

if (user[user\_id].u\_ofile[i] == SYSOPENFILE + 1)

{

user[user\_id].u\_uid = inode->di\_uid;

user[user\_id].u\_gid = inode->di\_gid;

for (j = 0; j < SYSOPENFILE; i++)

if (sys\_ofile[j].f\_count == 0)

{

user[user\_id].u\_ofile[i] = j;

sys\_ofile[j].f\_flag = (char)mode;

}

return i;

}

}

else

{

inode = ialloc();

di\_ith = iname(filename);

dir.size++;

dir.direct[di\_ith].d\_ino = inode->i\_ino;

inode->di\_mode = user[user\_id].u\_default\_mode;

inode->di\_uid = user[user\_id].u\_uid;

inode->di\_gid = user[user\_id].u\_gid;

inode->di\_size = 0;

inode->di\_number = 0;

for (i = 0; i < SYSOPENFILE; i++)

if (sys\_ofile[i].f\_count == 0)

{

break;

}

for (j = 0; j < NOFILE; i++)

if (user[user\_id].u\_ofile[j] == SYSOPENFILE + 1)

{

break;

}

user[user\_id].u\_ofile[j] = i;

sys\_ofile[i].f\_flag = (char)mode;

sys\_ofile[i].f\_count = 0;

sys\_ofile[i].f\_off = 0;

sys\_ofile[i].f\_inode = inode;

return j;

}

1. delete()用于删除文件，删除文件的算法流程图如图所示。下面是部分参考程序源代码：

图示

描述已自动生成

delete (char \*filename)

{

unsigned int dinodeid;

struct inode \*inode;

dinodeid = namei(filename);

if (dinodeid != (int)NULL)

inode = iget(dinodeid);

inode->di\_number--;

iput(inode);

}

1. read()函数用于读文件，读文件的算法流程图如图所示。下面是部分参考程序

图示

描述已自动生成

inode = sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_inode;

if (!(sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_flag & FREAD))

{

printf("\nthe file is not opened for read\n");

return 0;

}

temp\_buf = buf;

off = sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_off;

if ((off + size) > inode->di\_size)

size = inode->di\_size - off;

block\_off = off % BLOCKSIZ;

block = off / BLOCKSIZ;

if (block\_off + size < BLOCKSIZ)

{

fseek(fd, DATASTART + inode->di\_addr[block] \* BLOCKSIZ + block\_off, SEEK\_SET);

fread(buf, 1, size, fd);

return size;

}

fseek(fd, DATASTART + inode->di\_addr[block] \* BLOCKSIZ + block\_off, SEEK\_SET);

fread(temp\_buf, 1, BLOCKSIZ - block\_off, fd);

temp\_buf += BLOCKSIZ - block\_off;

j = (inode->di\_size - off - block\_off) / BLOCKSIZ;

for (i = 0; i < (int)(size - block\_off) / BLOCKSIZ; i++)

{

fseek(fd, DATASTART + inode->di\_addr[j + i] \* BLOCKSIZ, SEEK\_SET);

fread(temp\_buf, 1, BLOCKSIZ, fd);

temp\_buf += BLOCKSIZ;

}

1. )write()函数用于写操作，写文件的算法流程图如图所示。下面是部分参考程序源代码：

图示

描述已自动生成

i = user[user\_id].u\_ofile[cfd];

inode = sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_inode;

if (!(sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_flag & FWRITE))

{

printf("\nthe file is not opened for write\n");

return 0;

}

temp\_buf = buf;

off = sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_off;

block\_off = off % BLOCKSIZ;

block = off / BLOCKSIZ;

inode->di\_addr[block] = balloc();

if (block\_off + size < BLOCKSIZ)

{

fseek(fd, DATASTART + inode->di\_addr[block] \* BLOCKSIZ + block\_off, SEEK\_SET);

fwrite(buf, 1, size, fd);

return size;

}

fseek(fd, DATASTART + inode->di\_addr[block] \* BLOCKSIZ + block\_off, SEEK\_SET);

fwrite(temp\_buf, 1, BLOCKSIZ - block\_off, fd);

temp\_buf += BLOCKSIZ - block\_off;

for (i = 0; i < (int)(size - block\_off) / BLOCKSIZ - 1; i++)

{

inode->di\_addr[block + 1 + i] = balloc();

fseek(fd, DATASTART + inode->di\_addr[block + 1 + i] \* BLOCKSIZ, SEEK\_SET);

fwrite(temp\_buf, 1, BLOCKSIZ, fd);

temp\_buf += BLOCKSIZ;

}

// 下面是对不够一个block的数据进行处理

block\_off = (size - block\_off) % BLOCKSIZ;

block = inode->di\_addr[off + size / BLOCKSIZ + 1] = balloc();

fseek(fd, DATASTART + block \* BLOCKSIZ, SEEK\_SET);

fwrite(temp\_buf, 1, block\_off, fd);

sys\_ofile[user[user\_id].u\_ofile[cfd]].f\_off += size;

1. open()函数用于打开文件，打开文件的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

if (dinodeid != (int)NULL)

{

printf("\nThe file does not existed!!!\n");

return (int)NULL;

}

inode = iget(dinodeid);

if (!access(user\_id, inode, openmode))

{

printf("\nThe file was not allow to access!!!\n");

iput(inode);

return (int)NULL;

}

for (i = 1; i < SYSOPENFILE; i++)

if (sys\_ofile[i].f\_count == 0)

break;

if (i == SYSOPENFILE)

{

printf("\nsystem open file too much\n");

iput(inode);

return (int)NULL;

}

***4)目录操作***

1. 1s()函数用于浏览目录，浏览目录的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

for (i = 0; i < dir.size; i++)

{

if (dir.direct[i].d\_ino != DIEMPTY)

{

printf("%sDIRSIZ", dir.direct[i].d\_name);

temp\_inode = iget(dir.direct[i].d\_ino);

di\_mode = temp\_inode->di\_mode;

for (j = 0; j < 9; j++)

{

one = di\_mode % 2;

di\_mode = di\_mode / 2;

if (one)

printf("x");

else

printf("-");

}

if (temp\_inode->di\_mode && DIFILE == 1)

{

printf("%ld\n", temp\_inode->di\_size);

printf("block chain:");

for (i = 0; i < temp\_inode->di\_size / BLOCKSIZ + 1; i++)

printf("%4d", temp\_inode->di\_addr[i]);

printf("\n");

}

else

printf("<dir>\n");

iput(temp\_inode);

}

}

1. mkdir()函数用于创建目录，创建目录的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

dirid = namei(dirname);

if (dirid != -1) // dirid==-1表示没有该目录名存在；

{

inode = iget(dirid);

if (inode->di\_mode & DIDIR)

printf("\n%s directory already existed!!\n");

else

printf("\n%s is a file name&can not creat a dir the same name", dirname);

iput(inode);

return 0;

}

1. chdir()函数用于改变当前目录，修改目录的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

if (dirid == -1)

{

printf("\nThe %s is not a existed directory\n", dirname);

return 0;

}

inode = iget(dirid);

if (!access(user\_id, inode, user[user\_id].u\_default\_mode))

{

printf("\nThe filesystem has not access to the directory %s", dirname);

iput(inode);

return 0;

}

for (i = 0; i < dir.size; i++)

{

for (j = 0; j < DIRNUM; j++)

{

temp = dir.direct[j].d\_ino;

if (dir.direct[j].d\_ino == 0 || dir.direct[j].d\_ino > MAX)

break;

}

dir.direct[j].d\_ino = 0;

}

***5)用户的登录与注销操作***

1. login()函数用于用户登录，用户登录的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

for (i = 0; i < PWDNUM; i++)

{

if ((uid == pwd[i].p\_uid) && ((strcmp(passwd, pwd[i].password) == 0)))

{

j = 0;

while (j < USERNUM)

{

if (user[j].u\_uid == 0)

{

break;

}

else

{

j++;

}

}

if (j == USERNUM)

{

printf("\nThere are too much users in the filesystem, please wait!!\n");

return 0;

}

else

{

user[j].u\_uid = uid;

user[j].u\_gid = pwd[i].p\_gid;

user[j].u\_default\_mode = DEFAULTMODE;

}

break;

}

}

1. logout()函数用于注销用户，注销用户的算法流程图如图所示。部分参考程序源代码如下：

图示

描述已自动生成

for (i = 0; i < USERNUM; i++)

if (uid == user[i].u\_uid)

break;

if (i == USERNUM)

{

printf("\nThere is no such a user in filesystem.\n");

return (int)NULL;

}

for (j = 0; j < NOFILE; j++)

{

if (user[i].u\_ofile[j] != SYSOPENFILE + 1)

{

sys\_no = user[i].u\_ofile[j];

inode = sys\_ofile[sys\_no].f\_inode;

iput(inode);

sys\_ofile[sys\_no].f\_count--;

user[i].u\_ofile[j] = SYSOPENFILE + 1;

}

}

***6)文件系统的安装和退出***

1. install()函数用于安装文件系统，安装文件系统的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

printf("install\n");

fd = fopen("filesystem", "a+b");

if (fd == NULL)

{

printf("\nfilesys can not be loaded.\n");

exit(0);

}

fseek(fd, BLOCKSIZ, SEEK\_SET);

fread(&filsys, 1, sizeof(struct filsys), fd);

i = filsys.s\_free[49];

for (i = 0; i < NHINO; i++)

{

hinode[i].i\_forw = NULL;

}

1. halt()函数用于退出文件系统，退出文件系统的算法流程图如图所示。下面是部分参考程序源码：

图示

描述已自动生成

halt()

{

int i, j;

for (i = 0; i < USERNUM; i++)

{

if (user[i].u\_uid != 0)

{

for (j = 0; j < NOFILE; j++)

{

if (user[i].u\_ofile[j] != SYSOPENFILE + 1)

{

close(user[i].u\_ofile[j]);

user[i].u\_ofile[j] = SYSOPENFILE + 1;

}

}

}

}

fseek(fd, BLOCKSIZ, SEEK\_SET);

fwrite(&filsys, 1, sizeof(struct filsys), fd);

fclose(fd);

printf("\nGOOD bye. See You Next Time. Please turn off the switch.\n");

exit(0);

}

##### 2.2 EXT2 文件系统跟踪程序的设计与实现

# **2.2.1 程序设计思想**

程序ext2fs\_std将实现EXT2文件系统关键数据结构的显示任务，包括超级块信息、块组信息、块位图和inode 位图的信息、目录信息、inode信息以及显示文件内容。程序ext2fs\_std 紧跟的参数<device name>表示可用设备名，由于本程序不支持除了EXT2之外的其他文件系统，如果设备不是EXT2文件系统则会报错退出。然后通过程序提供的菜单选择用户所想了解的EXT2文件系统的数据结构中的信息，并且可以以只读方式访问普通文件以文本或者十六进制码显示。EXT2文件系统跟踪程序结构框图如图所示。

图示

描述已自动生成

# **2.2.2 程序实现**

***1) main()函数***

此函数的主要功能是在确认设备及文件系统正确后，打开并初始化超级块,然后显示菜单。可以根据自己的需要来选择使用其中的功能，流程图如图所示

图示

描述已自动生成

***2) super()函数***

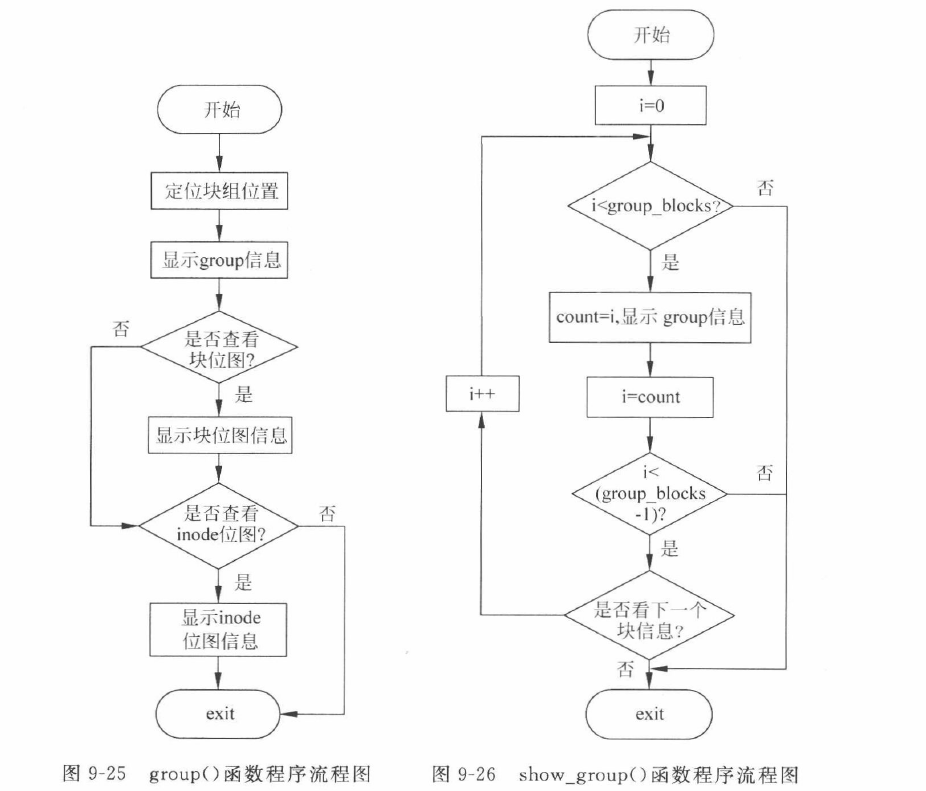
此函数的主要功能是显示超级块的数据结构中每个字段的信息，程序流程图如图所示。

图示

描述已自动生成

***3) show\_group()函数***

此函数的主要功能是显示每个块组的数据结构各字段的信息，并且可以根据需要查看块位图和inode位图的信息，程序流程图如图所示，下面为部分代码：



for (i = 0; i < group\_blocks; i++)

{

printf("\n\nGroup %d of %d", i, group\_blocks - 1);

count = i;

group(i);

i = count;

if (i < (group\_blocks - 1))

{

while (1)

{

printf("\n\nDo you want to view the next block? (y/n) ");

scanf("%s", &tmp);

str = (char \*)malloc(strlen(tmp));

strcpy(str, tmp);

if ((strcmp(str, "n") == 0) || (strcmp(str, "N") == 0) || (strcmp(str, "Y") == 0) || (strcmp(str, "y") == 0))

break;

}

if ((strcmp(str, rep1) == 0) || (strcmp(str, rep2) == 0))

break;

}

}

***4) directory()函数***

此函数的主要功能是根据输人的路径显示其目录下所有文件的内核目录数据结构字段信息，查找EXT2文件的一般步骤如下：

1. 读取超级块信息，了解磁盘分区的管理信息，特别是块长度和inode 表起始块号。
2. 找到根“/”目录信息。EXT2文件系统中，根目录的inode号是固定的，即EXT2ROOT INO=2，读取2号inode表信息，即可找到根目录所在的块号。在/include/Linux/ext2 fs.h中定义EXT2\_ROOT\_INO：

#define EXT2\_ROOT\_INO 2 /\*根 inode \*/

1. 读取根目录或普通目录所在块的信息，根据ext2\_dir\_entry\_2结构，找到子目录或文件的 inode号。
2. 读取子目录或文件的inode信息，找到文件数据所在的磁盘块号
3. 读取子目录或文件数据块的信息。如果是文件，则查找结束。如果是目录，转到(3)。EXT2文件数据块地址存放在指针数组i\_block[15]中。指针数组i\_block[0]至iblock[11]中直接包含文件块0~11的物理块地址，而i\_block[13]中包含一次间接寻址地址，i\_block[14]中包含二次间接寻址地址，i\_block[15]中包含三次间接寻址地址。程序流程图如图所示。

图示

描述已自动生成

部分代码如下：

while (search(ptr, ch) != -1)

{

if (i1 == 0)

{

ptr++;

continue;

}

else if (i1 > 0)

{

dir\_name = (char \*)malloc(i1 + 1);

n1 = 0;

while (n1 < i1)

{

dir\_name[n1] = \*ptr;

ptr++;

n1++;

}

dir\_name[n1] = '\0';

if (strlen(dir\_name) == strlen("root"))

{

if (strcmp(dir\_name, "root") == 0)

{

ptr++;

full\_dir\_name = ptr;

}

}

ptr++;

}

}

// x=find\_block\_no(2); /\*inode number 2 is reserved for root\*/

// x=x\*4096;

i1 = 0;

while (search(full\_dir\_name, ch) != -1)

{

if (i1 == 0)

{

full\_dir\_name++;

continue;

}

else if (i1 > 0)

{

dir\_name = (char \*)malloc(i1 + 1);

n1 = 0;

while (n1 < i1)

{

dir\_name[n1] = \*full\_dir\_name;

full\_dir\_name++;

n1++;

}

dir\_name[n1] = '\0';

full\_dir\_name++;

i1 = 0;

show();

if (file == 1)

{

printf("\n%s is not a directory\n", dir\_name);

file = 0;

return;

}

if (wrong\_name == 1)

{

wrong\_name = 0;

goto again;

}

}

}

***5) inode()函数***

此函数的主要功能是根据输人的文件的inode号显示其inode的数据结构各字段信息，并且可以根据需要以ASCII码或者十六进制码查看文件内容，程序流程图如图所示。

图示

描述已自动生成

以下是部分代码：

Inode = (void \*)(super\_block\_buffer);

printf("\nMode : %d", Inode->i\_mode); //&0xfff

printf("\tOctal : %o\t", Inode->i\_mode);

z = 6;

for (i = 0; i <= 6; i += 3)

{

temp1 = Inode->i\_mode & 0x1ff;

temp1 = temp1 >> z;

z = z - 3;

if (temp1 & 4)

printf("r");

else

printf("-");

if (temp1 & 2)

printf("w");

else

printf("-");

if (temp1 & 1)

printf("x");

else

printf("-");

}

printf("\n");

printf("\nUID : %01d", Inode->i\_uid);

printf("\nGID : %01d", Inode->i\_gid);

printf("\nSize : %9ld", Inode->i\_size);

printf("\nAccess time : %24s", ctime((time\_t \*)&Inode->i\_atime));

printf("\nCreation time : %24s", ctime((time\_t \*)&Inode->i\_ctime));

printf("\nModification time : %24s", ctime((time\_t \*)&Inode->i\_mtime));

printf("\nDeletion time : %24s", ctime((time\_t \*)&Inode->i\_dtime));

printf("\nLinks : %d", Inode->i\_links\_count);

printf("\nFlags : %03o", (Inode->i\_mode & 0x1ff000) >> 12);

printf("\nFLAGS : %ld", Inode->i\_flags);

if (S\_ISLNK(Inode->i\_mode))

{

printf("\n\nIt's a symbolic link.");

if (Inode->i\_size <= 60)

printf("\nPath conatined in Direct block is: %s", (char \*)&Inode->i\_block[0]);

else

printf("Slow Symbolic link");

return;

}

***6) Makefile 文件***

ext2fs\_std: main.o super.o group.o chdc.o inode.o file.o ascii.o blockbitmap.o inodebitmap.o

gcc -o ext2fs\_std main.o super.o group.o chdc.o inode.o file.o ascii.o blockbitmap.o inodebitmap.o

main.o: main.c ext2fs.h

gcc -c main.c

super.o: super.c ext2fs.h

gcc -c super.c

group.o: group.c ext2fs.h

gcc -c group.c

chdc.o: chdc.c ext2fs.h

gcc -c chdc.c

inode.o: inode.c ext2fs.h

gcc -c inode.c

file.o: file.c ext2fs.h

gcc -c file.c

ascii.o: ascii.c ext2fs.h

gcc -c ascii.c

blockbitmap.o: blockbitmap.c ext2fs.h

gcc -c blockbitmap.c

inodebitmap.o: inodebitmap.c ext2fs.h

gcc -c inodebitmap.c

clean:

rm \*.o

该makefile第一行指定ext2fs\_std由9个目标文件链接生成。第二行描述了如何从ext2fs\_std所依赖的文件建立可执行文件。接下来分别指定9个目标文件以及它们所依赖的.c和.h文件，和如何从目标所依赖的文件建立目标。最后两行是指定删除所有生成的.o文件的命令rm \*.o。

当.c或.h文件在编译之后又被修改，则make工具可自动重新编译生成.0，如果在前后两次编译之间.c和.h均没有被修改，而且.o还存在，就没有必要重新编译

# **第三章 实验结果分析**

##### 3.1 二级文件系统

# **3.1.1 程序说明**

本实验所提供的代码为C语言所编写。该程序模拟文件系统所提供的操作有login、logout、ls、mkdir、chdir 和create(即创建一个文件)等。程序所提供的功能通过主程序的菜单选择来实现。

代码内已预置了5个用户，其用用户ID和口令分别为

用户ID

2116

2117

2118

2119

2220

口令

don1

don2

abcd

don4

don5

当然你也可以使用PaperYes论文排版，10秒搞定，安全高效！

# **3.1.2 登录**

输入用户ID和密码，成功登录

文本

描述已自动生成

# **3.1.3 创建目录**

创建一个新目录，名为dirr

文本

描述已自动生成

# **3.1.4 创建文件**

创建一个新文件，名为file1

文本

描述已自动生成

# **3.1.5 浏览目录**

查看当前文件夹内的所有文件

文本

描述已自动生成

可以看到刚刚创建的文件file1

# **3.1.6 切换目录**

切换到刚刚创建的目录dirr

文本

描述已自动生成

再在dirr下创建一个新文件dd

文本

描述已自动生成

此时浏览目录

文本

描述已自动生成

能看到创建的新文件dd

# **3.1.7 登出**

退出系统

文本

描述已自动生成

##### 3.2 ET2文件系统

# **3.2.1 开始**

运行编译好的程序ext2fs\_std，注意这是一个32位的程序，所以需要安装32位库。/lib/ld-linux.so.2 文件是一个ELF动态链接器/加载器，是glibc包的一部分。可以使用以下命令进行安装：

***$ sudo yum install glibc.i686***

程序的用法是

***./ext2fs\_std <device name>***

对于device name，可以这样查看，使用以下命令：

***lsblk***

根据 lsblk 命令的输出来看，我的设备名应该是 vda。这个设备下有一个分区，分区的名字是 vda1，它被挂载在根目录(/)

使用以下命令运行：

***./ext2fs\_std /dev/vda1***

文本

描述已自动生成

# **3.2.2 超级块数据结构显示**

当输入命令s将显示如图所示的信息，图中的Magic Signature是文件系统魔数是用来辨别文件系统版本的标识，EF53代表的就是EXT2文件系统。

文本

描述已自动生成

# **3.2.3 文件系统块位图显示**

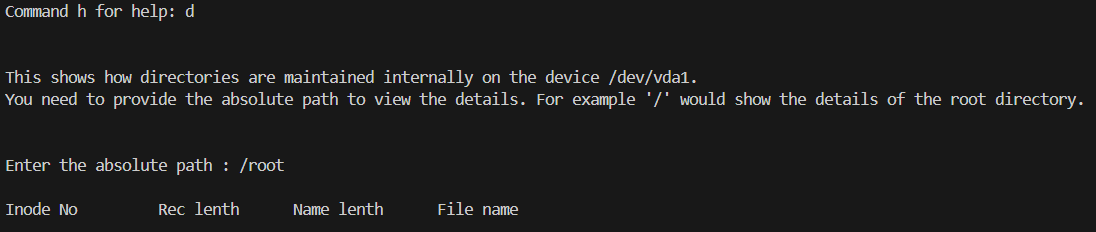
当输入命令g将显示如图所示的信息，并且在提示后输人y查看部分块位图，按回车键翻页直到本块块位图全部显示，如果要继续查看inode位图输人y，不再查看输人n。块位图中显示1表示数据块已分配，0表示数据块未分配。inode位图的1和0表示索引节点的分配情况。

文本

描述已自动生成

# **3.2.4 路径显示**

当输入命令d并且输人所要查看的路径，将显示如图所示的目录信息。



参数说明:

Inode-索引节点号

rec\_len-目录项长度

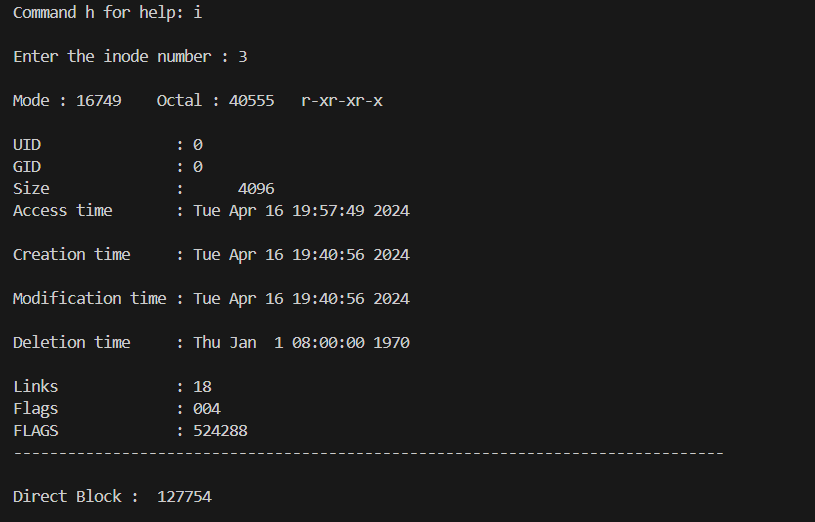
name\_len一文件名长度

name一文件名。

当然你也可以使用PaperYes论文排版，10秒搞定，安全高效！

# **3.2.5 文件内容显示**

当输人命令i，然后输入所要查看文件inode号，将显示如图所示的信息。如果要查看文件内容，可以在提示处输入y，其中普通文件的权限有3部分：读、写和执行。分别用r、w、x来表示。目录也有这3种表示方式，相应的权限为列出目录内容、在目录中建立或删除文件、进入和退出目录。本例中所示文件的权限为r-xr-xr-x，这9个字符每3个一组依次代表文件的所有者、所有者所在用户组和组外其他用户对该文件的访问权限，当用户没有相应的权限时，系统在该权限的对应位置上用“-”表示。



##### 参考文献

[1] 附件1-1-1 操作系统实验指导——基于Linux内核.