**操作系统实验班大作业**

ext2的用户态模拟

**蒋捷 / 1200012708 & 兰兆千 / 1100012458 & 邢曜鹏 / 1200012835 &**

**赵万荣 / 1200012808 & 周昊宇 / 1200012823 （音序）**

# 概要

作为本小组的完成项目之一，我们使用C语言模仿ext2实现了一个简洁的、操作系统无关的文件系统模拟器，包含格式化、显示文件（目录）、创建文件、用户管理等功能，而且能模拟超级块的读写、节点的读写。这是一个比真实文件系统简单得多，但又能基本体现文件系统思想的程序。

由于该文件系统是一个模拟器，所以运行环境并不局限于Linux下，且不同于真实的文件系统，会创建一个二进制文件代表文件系统，其中存有用户信息、节点信息、超级块信息，其行为都是通过标准库中的文件读写函数来模拟。

## 二进制文件的内容布局：

|  |  |
| --- | --- |
| 起始地址 | 内容 |
| 0x000000 |  |
| 0x000200 | 超级块 |
| 0x000400 | inode #0 |
| 0x000420 | inode #1 |
| 0x000440 | inode #2 |
| …… | …… |
| 0x004400 | block #0(/) |
| 0x004600 | block #1(etc) |
| 0x004800 | block #2(passwd) |
| …… | …… |
| 0x039c00 | unused |
| 0x044400 | 文件结束 |

# 使用说明

## **文件系统所支持的操作**

login（登录）、logout（登出）、ls（浏览目录）、mkdir（创建目录）、chdir（更改当前目录）和create（创建文件）。

## 登录用户说明

在format.h中内置了5个用户，其用户ID和密码分别为

|  |  |
| --- | --- |
| 用户ID | 密码 |
| 2116 | don1 |
| 2117 | don2 |
| 2118 | abcd |
| 2119 | don4 |
| 2220 | don5 |

# 实现

## 数据结构

### 索引节点的数据结构

struct inode

{

struct inode \*i\_forw; // 指向前一个inode

struct inode \*i\_back; // 指向后一个inode

char i\_flag; // 标志

unsigned int i\_ino; // inode编号

unsigned int i\_count; // inode引用次数

unsigned int di\_addr[NADDR]; // 数据块地址

unsigned short di\_number; // 对应的目录数

unsigned short di\_mode; // 权限位

unsigned short di\_uid; // 所属用户编号

unsigned short di\_gid; // 所属用户组编号

unsigned short di\_size; // 大小

};

### 超级块数据结构

struct filsys

{

unsigned short s\_isize; // 对应inode大小

unsigned long s\_fsize; // 对应超级块大小

unsigned int s\_nfree; // 指向空闲块的指针

unsigned short s\_pfree;

unsigned int s\_free[NICFREE]; // 空闲块数组

unsigned int s\_ninode;

unsigned short s\_pinode;

unsigned int s\_inode[NICINOD];

unsigned int s\_rinode;

char s\_fmod; // 权限位

### 目录相关数据结构

struct dinode // 对应inode里参数

{

  unsigned short di\_number;

unsigned short di\_mode;

unsigned short di\_uid;

unsigned short di\_gid;

unsigned long di\_size;

unsigned int di\_addr[NADDR];

};

struct direct

{

char d\_name[DIRSIZ];

unsigned int d\_ino; // 对应inode编号

};

struct dir

{

struct direct direct[DIRNUM];

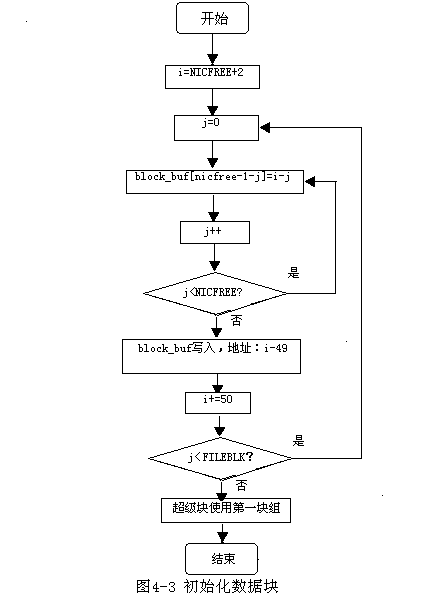
int size;

};

## 功能实现（限于篇幅就不放代码了）

### 文件系统的初始化

初始化数据块的函数位于format.h，流程图如下。



NICFREE为每个块组的大小，默认为50；FILEBLK为系统允许最多块数，默认为512。

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。

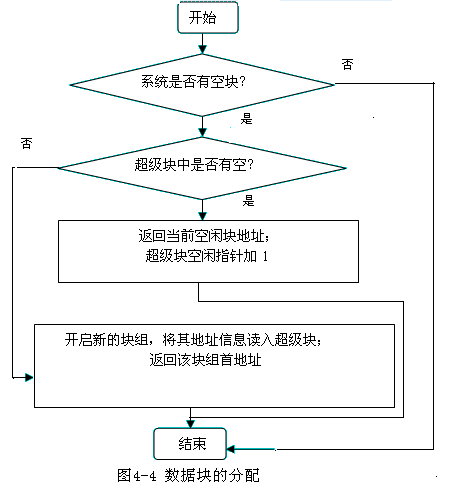
当前使用块组的各个数据块地址存放在全局变量block\_buf[]当中。

### 数据块的分配和回收

数据块的分配和回收由ballfre.h的两个函数完成。

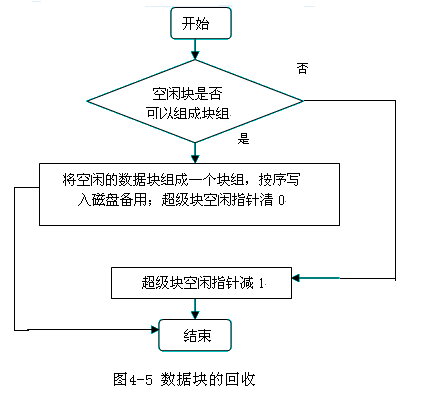
1. balloc：分配数据块

流程图如下。



1. bfree：回收数据块

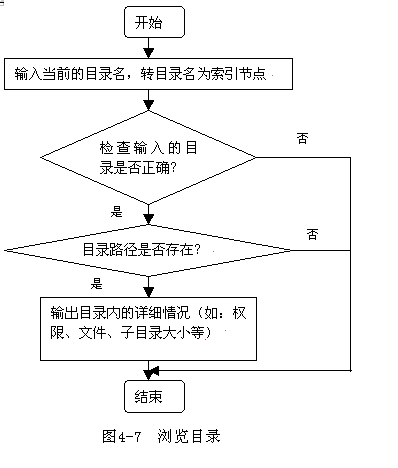
流程图如下。



### 目录操作

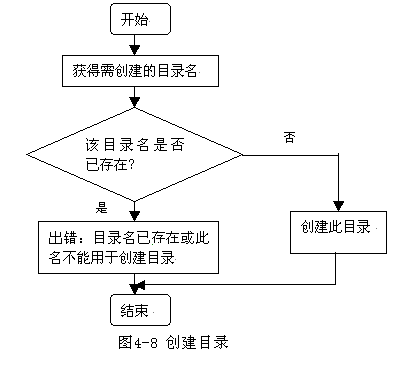
1. 浏览目录：dir()

流程图如下。



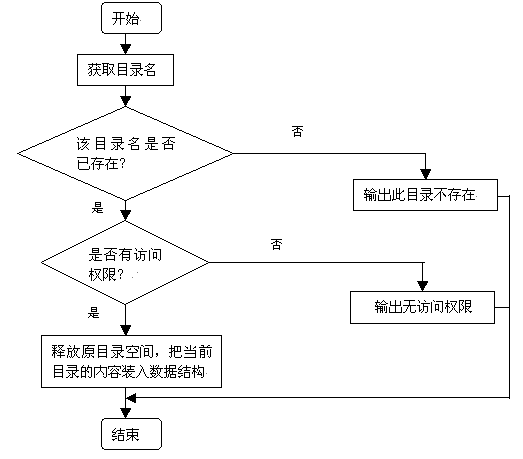
1. 创建目录：mkdir()

流程图如下。



1. 改变当前目录：chdir()

流程图如下。



# 总结与感想

一个完整的文件系统是非常复杂的。我们实现的只是一个具有文件系统基本行为的模拟器，然而在编写过程中也能体会到文件系统的基本思想。与在内核中编写真正的文件系统不同，我们不需要翻找内核代码寻找层层包装的函数，不用考虑到微不可察的同步问题，不用实现几十种文件系统的接口，也不用和真实设备打交道——这些也并不是文件系统的重点所在。通过完成本项目，我们亲手打造的“文件系统”让我们对文件系统相关的知识、ext2的布局与块管理策略有了更深入的理解。