**目录**

[**一、 选题背景** 3](#_Toc1390461)

[**二、 方案论证(设计理念)** 3](#_Toc1390462)

[**三、 过程论述** 3](#_Toc1390463)

[**四、 结果分析** 3](#_Toc1390464)

[**五、 课程设计总结** 3](#_Toc1390465)

**模拟Linux文件系统**

1. **选题背景**

**课程设计目的**

本设计的目的是实现操作系统和相关系统软件的设计，其中涉及进程编程、I/O操作、存储管理、文件系统等操作系统概念。

**课程设计要求**

设计任务：模拟Linux文件系统

在任一OS下，建立一个大文件，把它假象成一张盘，在其中实现一个简单的模拟Linux文件系统。

1. 在现有机器硬盘上开辟100M的硬盘空间，作为设定的硬盘空间。
2. 编写一管理程序simdisk对此空间进行管理，以模拟Linux文件系统，要求：
3. 盘块大小1k
4. 空闲盘块的管理：Linux位图法
5. 结构：超级块, i结点区, 根目录区
6. 该simdisk管理程序的功能要求如下：
   1. info: 显示整个系统信息(参考Linux文件系统的系统信息)，文件可以根据用户进行读写保护。目录名和文件名支持全路径名和相对路径名，路径名各分量间用“/”隔开。
   2. cd …: 改变目录：改变当前工作目录，目录不存在时给出出错信息。
   3. dir …: 显示目录：显示指定目录下或当前目录下的信息，包括文件名、物理地址、保护码、文件长度、子目录等（带/s参数的dir命令，显示所有子目录）。
   4. md …: 创建目录：在指定路径或当前路径下创建指定目录。重名时给出错信息。
   5. rd …: 删除目录：删除指定目录下所有文件和子目录。要删目录不空时，要给出提示是否要删除。
   6. newfile …: 建立文件。
   7. cat …: 打开文件。
   8. copy …: 拷贝文件，除支持模拟Linux文件系统内部的文件拷贝外，还支持host文件系统与模拟Linux文件系统间的文件拷贝，host文件系统的文件命名为<host>…，如：将windows下D：盘的文件\data\sample\test.txt文件拷贝到模拟Linux文件系统中的/test/data目录，windows下D：盘的当前目录为D：\data，则使用命令：

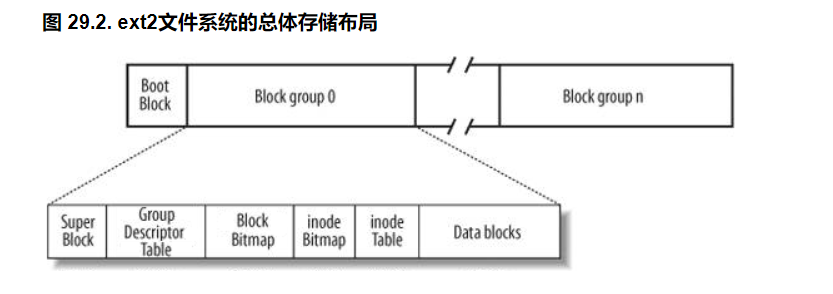
simdisk copy <host>D：\data\sample\test.txt /test/data

或者：simdisk copy <host>D：sample\test.txt /test/data

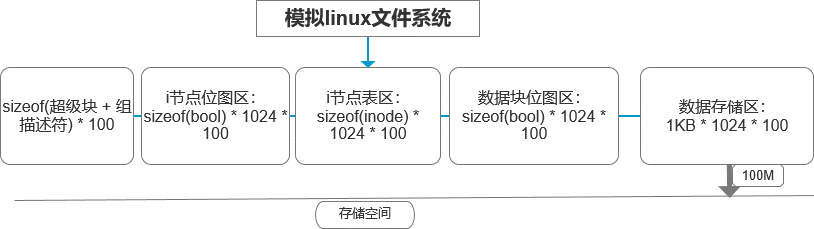
* 1. del …: 删除文件：删除指定文件，不存在时给出出错信息。
  2. check: 检测并恢复文件系统：对文件系统中的数据一致性进行检测，并自动根据文件系统的结构和信息进行数据再整理。

1. 程序的总体流程为：
   1. 初始化文件目录；
   2. 输出提示符，等待接受命令，分析键入的命令；
   3. 对合法的命令，执行相应的处理程序，否则输出错误信息，继续等待新命令，直到键入EXIT退出为止。
2. **方案论证(设计理念)**

本系统使用了ext2文件系统的结构形式，但磁盘存储序列上却有不同；



本系统的结构包括**超级块、组描述符、i节点位图、i节点表、数据块位图和数据存储区**这几个部分组成，将整个文件系统空间分为100个数据块组，但是每个数据块组并不是像ext2文件系统那样组与组之间隔开，而是每个结构全部**连续存储**完之后再存储下一个，所以整个文件系统的磁盘空间分布如下，



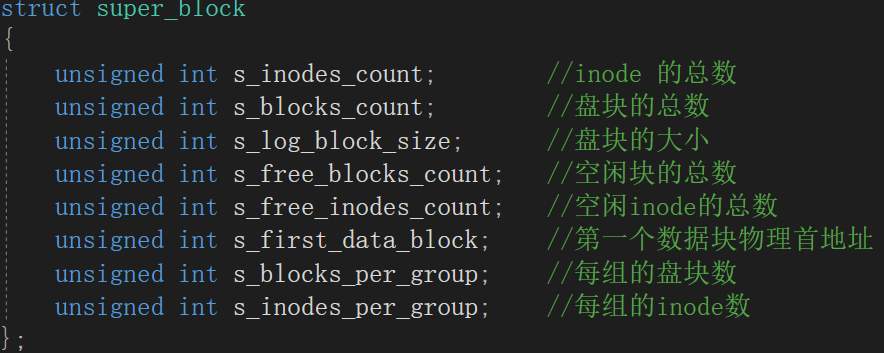
100 指的是文件系统共有100个数据块组

1024 指的是每个数据块组中都包含1024个i节点和1024个数据块

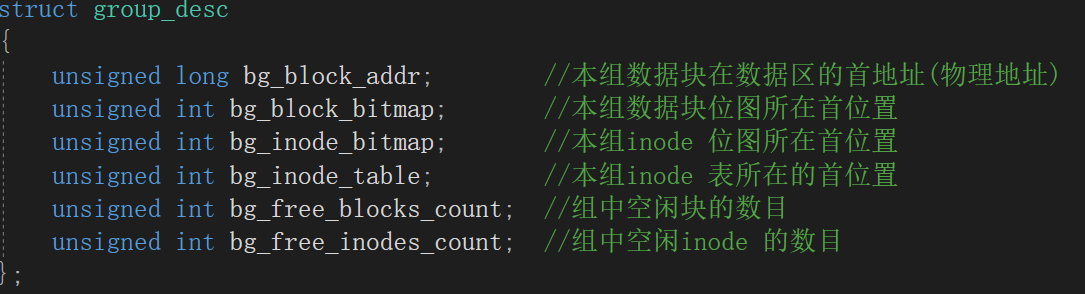
每个数据块大小是1KB，所以仅数据存储区域就占了100M内存

**数据结构**如下所示：

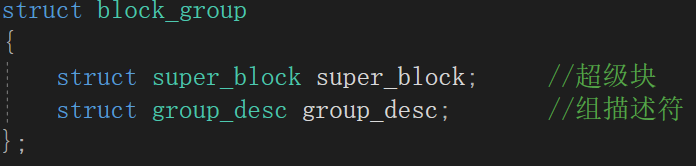
**超级块：**记录整个系统的重要数据，每个数据块组都有一个超级块，但是内容都一样



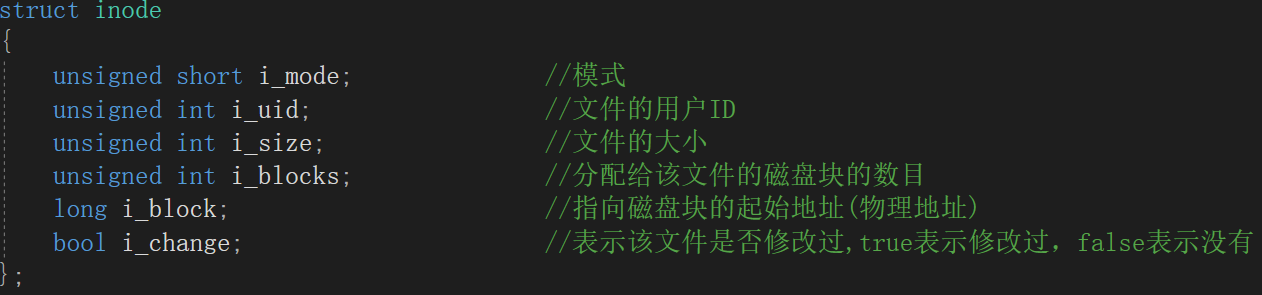
**组描述符：**描述本个数据块组的重要信息



**超级块 + 组描述符**



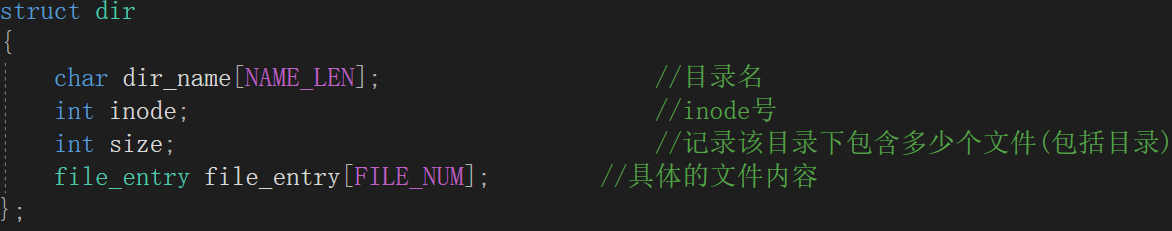
**i节点：**描述文件(普通文件+目录)的属性，每个文件就是一个i节点



**文件(普通文件或目录)**



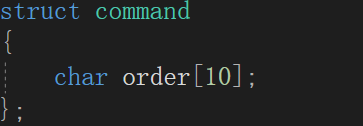
**具体目录:** 用于存储指定具体目录所包含文件(包括目录)的数目和具体的文件内容(即file\_entry)



**用户：**权限、ID、密码

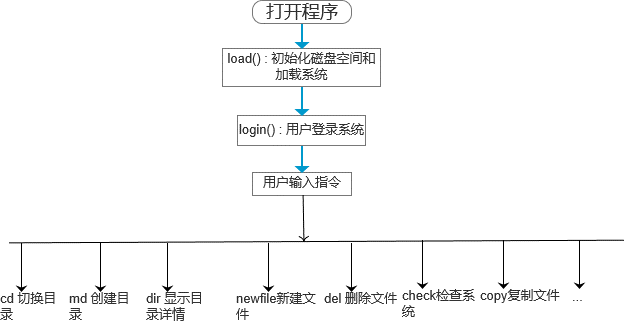


**输入指令：**用户输入的指令

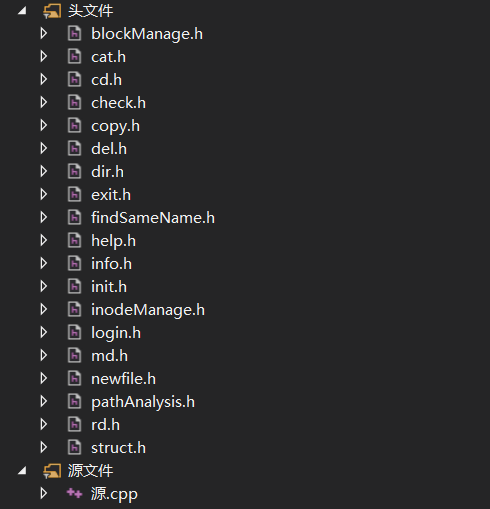


1. **过程论述**

整个程序执行过程如下

**模拟Linux文件系统执行流程**

**工程目录**



**主要函数实现**

**load():**初始化磁盘空间和加载文件系统，将文件系统中的信息读入内存

/\*

\*功能 : 初始化磁盘空间和加载文件系统

\*/

void load()

{

//初始化命令

strcpy(cmd[0].order, "init");

strcpy(cmd[1].order, "info");

strcpy(cmd[2].order, "cd");

strcpy(cmd[3].order, "dir");

strcpy(cmd[4].order, "md");

strcpy(cmd[5].order, "rd");

strcpy(cmd[6].order, "newfile");

strcpy(cmd[7].order, "cat");

strcpy(cmd[8].order, "copy");

strcpy(cmd[9].order, "del");

strcpy(cmd[10].order, "check");

strcpy(cmd[11].order, "exit");

strcpy(cmd[12].order, "help");

//初始化路径

strcpy(current\_path, "root>");

//注册用户

\_user[0].mode = ADMIN;

\_user[0].uid = 201630599319;

strcpy(\_user[0].password, "599319");

fd = fopen("LINUX\_FILE\_SYSTEM", "rb");

if (fd == NULL)

{

fflush(stdin);

initialize();

return;

}

// 读入文件系统中内容到内存

fread(\_block\_group, sizeof(block\_group), BLOCKS\_GROUP\_NUM, fd);//读入超级块和组描述符

fread(inode\_bitmap, sizeof(bool), TOTAL\_INODES\_NUM, fd); //读入inode位图

fread(inode\_table, sizeof(inode), TOTAL\_INODES\_NUM, fd); //读入inode表

fread(block\_bitmap, sizeof(bool), TOTAL\_BLOCKS\_NUM, fd); //读入数据块位图

fread(&\_current\_dir, sizeof(dir), 1, fd); //读入根目录

}

**initialize ():**创建磁盘空间并格式化系统

/\*

\*功能:初始化文件系统

\*/

void initialize()

{

printf("正在进行磁盘划分和初始化系统...\n");

int i;

//分配空间

long file\_size = FIRST\_DATA\_ADDR + BLOCK\_SIZE \* TOTAL\_BLOCKS\_NUM;

char \*buf = new char[file\_size];

//写入磁盘

fd = fopen("LINUX\_FILE\_SYSTEM", "wb+");

if (fd == NULL)

{

printf("Linux\_file\_system create failed !\n");

system("pause");

}

fseek(fd, 0, SEEK\_SET);

fwrite(buf, file\_size, 1, fd);

fclose(fd);

delete(buf);

//初始化数据块组

for (i = 0; i < BLOCKS\_GROUP\_NUM; i++)

{

//初始化超级块

\_block\_group[i].super\_block.s\_inodes\_count = TOTAL\_INODES\_NUM;

\_block\_group[i].super\_block.s\_blocks\_count = TOTAL\_BLOCKS\_NUM;

\_block\_group[i].super\_block.s\_log\_block\_size = BLOCK\_SIZE;

\_block\_group[i].super\_block.s\_free\_blocks\_count = TOTAL\_BLOCKS\_NUM - DIR\_SIZE; //一个用于根目录

\_block\_group[i].super\_block.s\_free\_inodes\_count = TOTAL\_INODES\_NUM - 1; //一个用于根目录

\_block\_group[i].super\_block.s\_first\_data\_block = FIRST\_DATA\_ADDR;

\_block\_group[i].super\_block.s\_blocks\_per\_group = BLOCKS\_PER\_GROUP;

\_block\_group[i].super\_block.s\_inodes\_per\_group = INODES\_PER\_GROUP;

//初始化组描述符

\_block\_group[i].group\_desc.bg\_block\_addr = FIRST\_DATA\_ADDR + i \* BLOCKS\_PER\_GROUP\*BLOCK\_SIZE;

\_block\_group[i].group\_desc.bg\_block\_bitmap = i \* BLOCKS\_PER\_GROUP;

\_block\_group[i].group\_desc.bg\_inode\_bitmap = i \* INODES\_PER\_GROUP;

\_block\_group[i].group\_desc.bg\_inode\_table = i \* INODES\_PER\_GROUP;

\_block\_group[i].group\_desc.bg\_free\_blocks\_count = BLOCKS\_PER\_GROUP;

\_block\_group[i].group\_desc.bg\_free\_inodes\_count = INODES\_PER\_GROUP;

}

//初始化数据块位图,所有数据块均未被使用

for (i = 0; i < TOTAL\_BLOCKS\_NUM; i++)

block\_bitmap[i] = NOT\_USED;

//初始化inode位图,所有inode均未被使用

for (i = 0; i < TOTAL\_INODES\_NUM; i++)

inode\_bitmap[i] = NOT\_USED;

//初始化inode表

for (i = 0; i < TOTAL\_INODES\_NUM; i++)

{

inode\_table[i].i\_mode = READ\_ONLY;

inode\_table[i].i\_uid = 0;

inode\_table[i].i\_size = 0;

inode\_table[i].i\_blocks = 0;

inode\_table[i].i\_block = -1;

inode\_table[i].i\_change = false;

}

/\*设置第0个数据块为根目录的一系列信息\*/

//初始化根目录所在数据块组的组描述符

\_block\_group[0].group\_desc.bg\_free\_blocks\_count = BLOCKS\_PER\_GROUP - DIR\_SIZE; //一个用于根目录

\_block\_group[0].group\_desc.bg\_free\_inodes\_count = INODES\_PER\_GROUP - 1; //一个用于根目录

//初始化根目录的数据块位图

for (i = 0; i < DIR\_SIZE; i++)

block\_bitmap[i] = USED;

//初始化根目录的inode 位图

inode\_bitmap[0] = USED;

//初始化根目录的inode

inode\_table[0].i\_mode = \_DIRECTORY; //模式为目录

inode\_table[0].i\_uid = 201630599319; //管理员的uid

inode\_table[0].i\_size = sizeof(dir); //根目录的大小

inode\_table[0].i\_blocks = DIR\_SIZE; //一个目录分配DIR\_SIZE个数据块

inode\_table[0].i\_block = FIRST\_DATA\_ADDR; //根目录在文件的起始位置

inode\_table[0].i\_change = false;

//往根目录项中添加内容

strcpy(\_current\_dir.dir\_name, "root"); //根目录名为root

\_current\_dir.inode = 0; //根目录的inode 号为0

\_current\_dir.size = 2; //一个当前目录，一个上一级目录

strcpy(\_current\_dir.file\_entry[0].name, "."); //当前目录

\_current\_dir.file\_entry[0].inode = 0; //即还是根目录

strcpy(\_current\_dir.file\_entry[1].name, ".."); //上一级目录

\_current\_dir.file\_entry[1].inode = 0; //根目录的上一级目录还是自己

//内存中的数据写入文件

fd = fopen("LINUX\_FILE\_SYSTEM", "rb+");

if (fd == NULL)

{

printf("Linux\_file\_system create failed !\n");

system("pause");

}

fseek(fd, 0, SEEK\_SET);

fwrite(&\_block\_group[0], sizeof(block\_group), BLOCKS\_GROUP\_NUM, fd); //把数据块组写到文件上

fwrite(&inode\_bitmap[0], sizeof(bool), TOTAL\_INODES\_NUM, fd); //把inode 位图写到文件上

fwrite(&inode\_table[0], sizeof(inode), TOTAL\_INODES\_NUM, fd); //把inode 表写到文件上

fwrite(&block\_bitmap[0], sizeof(bool), TOTAL\_BLOCKS\_NUM, fd); //把数据块位图写到文件上

fwrite(&\_current\_dir, sizeof(dir), 1, fd); //把根目录写到文件上

fclose(fd);

}

**info():显示系统信息**

/\*

\*显示系统信息，包括总盘块数，盘块大小，每组盘块数，空闲块数等

\*/

void info()

{

printf("\n%15s\n","模拟Linux文件系统信息:");

printf("%15s %10d %-3s\n","盘块的大小:",\_block\_group[0].super\_block.s\_log\_block\_size,"字节");

printf("%15s %10d %-3s\n","每组的盘块数:",\_block\_group[0].super\_block.s\_blocks\_per\_group,"块");

printf("%15s %10d %-3s\n","总的盘块数:",\_block\_group[0].super\_block.s\_blocks\_count,"块");

printf("%15s %10d %-3s\n","空闲块的总数:",\_block\_group[0].super\_block.s\_free\_blocks\_count,"块");

printf("%15s %10d %-3s\n","总容量为:",\_block\_group[0].super\_block.s\_blocks\_count\*BLOCK\_SIZE,"字节");

printf("%15s %10d %-3s\n","剩余空间为:",\_block\_group[0].super\_block.s\_free\_blocks\_count\*BLOCK\_SIZE,"字节");

}

**cd()**

/\*功能：改变当前的工作目录

\*

\*实现：1、若路径形式为" ../目录 "，则先返回上一级目录后再处理

\* 2、若路径形式为" ./目录 "，则直接在本级目录处理

\* 3、若路径形式为" root/目录 "，则返回根目录后再处理

\* 4、若路径形式为" 目录/第一级目录 "，则直接在本级目录处理

\*

\*path : 用户所输入的路径

\*/

void cd(char \*path)

{

dir temp;

if (getLastDir(path, temp))

\_current\_dir = temp;

else {

printf("目录切换失败!\n");

return;

}

findFullPath(\_current\_dir);

}

**dir : dirDisplay()**

/\*

\*功能:显示目录,显示当前目录或指定目录(绝对路径或相对路径)

\*

\*实现：1、若路径形式为" ../目录 "，则先返回上一级目录后再处理

\* 2、若路径形式为" ./目录 "，则直接在本级目录处理

\* 3、若路径形式为" root/目录 "，则返回根目录后再处理

\* 4、若路径形式为" 目录/第一级目录 "，则直接在本级目录处理

\*

\*path : 用户输入的目录

\*/

void dirDisplay(char \*path)

{

dir temp;

if (getLastDir(path, temp))

{

printf("%8s 的目录\n\n", path);

display(temp);

}

else

printf("目录显示失败!\n");

}

**copy()**

/\*

\*功能:拷贝文件

\*

\*算法步骤:首先检查给出的两个路径是否正确；

\* 如果正确则找到对应的文件，读出内容到内容缓冲区中，再将内容缓冲区的内容写到目的路径中

\* 否则提示错误信息

\*src\_path : 源文件所在路径

\*des\_path ： 拷贝文件至此路径中

\*/

void copy\_file(char\*src\_path,char\*des\_path)

{

dir temp\_dir; //保存目的目录

char file\_name[NAME\_LEN]; //文件名

char \*buf; //文件内容缓冲区

long len = 0; //文件长度

char divide; //保存分隔符

if(isHostFile(src\_path)) //从host文件系统拷贝到模拟Linux文件系统

{

fd = fopen(src\_path,"rb");

if(fd == NULL)

{

printf("文件不存在,文件拷贝失败!\n");

return;

}

fseek(fd,0,SEEK\_END);

len = ftell(fd); // 获取fd指针的当前位置

buf = new char[len]; //申请文件内容缓冲区

buf[len-1] = 0;

fseek(fd,0,SEEK\_SET);

fread(buf,len-1,1,fd); //读取文件的内容

fclose(fd);

divide = '\\'; //windows路径分隔符

strcpy(file\_name,strrchr(src\_path,divide) +1); //获得文件名

if(getLastDir(des\_path,temp\_dir)) //找到目的目录

{

write\_back(temp\_dir,file\_name,buf,len);

return;

}

else

printf("文件拷贝失败!\n");

}

else

{

if(isHostFile(des\_path)) //从模拟Linux文件系统拷贝到host文件系统

{

if(dividePathAndName(src\_path,file\_name,temp\_dir))

{

int inode; // 源文件的i节点号

for(int i=2;i<temp\_dir.size;i++)

if(strcmp(temp\_dir.file\_entry[i].name,file\_name)==0)

inode = temp\_dir.file\_entry[i].inode;

buf = new char[inode\_table[inode].i\_size-1]; //分配缓冲区

len = read\_file(temp\_dir,file\_name,inode,buf);

char \*complete\_path = new char[MAX\_PATH\_LEN]; //Windows中的完整路径

sprintf(complete\_path,"%s\\%s",des\_path,file\_name);

fd = fopen(complete\_path,"wb+");

if(fd == NULL)

{

printf("文件拷贝失败!\n");

delete(buf);

delete(complete\_path);

return;

}

fwrite(buf,len-1,1,fd);

fclose(fd);

delete(complete\_path);

delete(buf);

return;

}

printf("文件拷贝失败!\n");

}

else //Linux文件系统内部拷贝

{

if(dividePathAndName(src\_path,file\_name, temp\_dir))

{

int f\_inode;

for(int i=2;i<temp\_dir.size;i++)

{

if(strcmp(temp\_dir.file\_entry[i].name,file\_name)==0 && inode\_table[temp\_dir.file\_entry[i].inode].i\_mode == \_FILE)

{

f\_inode = temp\_dir.file\_entry[i].inode;

break;

}

}

buf = new char[inode\_table[f\_inode].i\_size]; //分配缓冲区

len = read\_file(temp\_dir,file\_name,f\_inode,buf);

char \*complete\_path = new char[MAX\_PATH\_LEN]; //Linux中的完整路径

sprintf(complete\_path,"%s/%s",des\_path,file\_name);

if(dividePathAndName(complete\_path,file\_name, temp\_dir))

write\_back(temp\_dir,file\_name,buf,len);

else

printf("文件拷贝失败!\n");

delete(complete\_path);

delete(buf);

}

}

}

}

**newfile()**

/\*

\*功能:建立文件

\*

\*实现:同样先检查路径，然后进行系列能否创建文件的判断，可建后修改相应的磁盘信息

\*

\*path : 用户输入的路径

\*/

void newfile(char \*path)

{

dir temp\_dir;

char dirName[NAME\_LEN];

if (dividePathAndName(path, dirName, temp\_dir))

create\_file(temp\_dir, dirName);

else

printf("文件创建失败!\n");

}

**create\_file()**

/\*

\*功能:在给定的目录temp\_dir下创建文件名为file\_name的文件

\*

\*算法步骤:检查temp\_dir下是否已经存在名为file\_name的文件，如果有则创建失败

\* 否则输入文件内容，直到输入回车，分配数据块和inode，如果分配成功则，修改超级块、

\* 组描述符、数据块位图、inode位图、文件所在的目录、数据区等，否则提示错误信息

\*temp\_dir: 在此目录下建立文件

\*file\_name : 文件的名字

\*/

void create\_file(dir temp\_dir, char \*file\_name)

{

//检查是否有同名文件

if (findSameName(temp\_dir, file\_name))

{

printf("%s 目录下已存在名为 %s 的文件，文件创建失败!\n", temp\_dir.dir\_name, file\_name);

return;

}

//读取文件内容

char content; //文件内容中的单个字符

int i, size = 5; //i为循环控制变量，size为文件内容buffer的大小，当输入大于size时，size成倍增加

char \*buffer = new char[size]; //用于存取文件内容，随size增加而增大

char \*temp; //用于当buffer变大时，用于临时存取buffer的内容

int count = 0; //用于统计读取的文件内容的大小

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

buffer[i] = 0;

printf("请输入文件内容，按 (# + Enter) 结束输入:\n\n");

while ((content = getchar()) != '#') //直到输入回车结束

{

buffer[count] = content;

count++;

if (count >= size - 1) //当输入内容超过缓冲区大小时，缓冲区大小翻倍

{

temp = new char[size];

strcpy(temp, buffer); //将原来缓冲区内容保存到temp中

delete(buffer);

size = size \* 2; //文件内容缓冲区成倍增加

buffer = new char[size];//重新分配缓冲区大小

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

buffer[i] = 0;

strcpy(buffer, temp); //将原来的内容放回缓冲区

delete(temp);

}

}

//分配文件数据块空间和inode

int block\_num = 0; //存储文件需要的数据块数目

long addr = -1; //存储文件的数据块首地址

int b\_bitmap\_index = -1;//存储文件的数据块在数据块位图中的首地址

int f\_inode = -1;

if ((count + 1) % BLOCK\_SIZE == 0)

block\_num = (count + 1) / BLOCK\_SIZE;

else

block\_num = (count + 1) / BLOCK\_SIZE + 1;

addr = getBlock(block\_num, &b\_bitmap\_index);

if (addr < 0)

{

printf("文件太大，硬盘空间不够，文件创建失败!\n");

return;

}

f\_inode = getInode();

//更新文件inode内容

inode\_table[f\_inode].i\_block = addr;

inode\_table[f\_inode].i\_blocks = block\_num;

inode\_table[f\_inode].i\_change = false;

inode\_table[f\_inode].i\_mode = \_FILE;

inode\_table[f\_inode].i\_size = count + 1;

inode\_table[f\_inode].i\_uid = uid;

//更新文件所在目录的内容

temp\_dir.file\_entry[temp\_dir.size].inode = f\_inode;

strcpy(temp\_dir.file\_entry[temp\_dir.size].name, file\_name);

temp\_dir.size++;

if (temp\_dir.inode == \_current\_dir.inode)

\_current\_dir = temp\_dir;

//写回硬盘

fd = fopen("LINUX\_FILE\_SYSTEM", "rb+");

if (fd == NULL)

{

printf("Linux file system file creat failed !\n");

exit(0);

}

fseek(fd, 0, SEEK\_SET);

fwrite(&\_block\_group[0], sizeof(block\_group), BLOCKS\_GROUP\_NUM, fd); //把数据块组写回文件

fseek(fd, (sizeof(block\_group)\*BLOCKS\_GROUP\_NUM + sizeof(bool)\*f\_inode), SEEK\_SET);

fwrite(&inode\_bitmap[f\_inode], sizeof(bool), 1, fd); //把inode 位图写回文件

fseek(fd, (sizeof(block\_group)\*BLOCKS\_GROUP\_NUM + sizeof(bool)\*TOTAL\_INODES\_NUM

+ sizeof(inode)\*f\_inode), SEEK\_SET);

fwrite(&inode\_table[f\_inode], sizeof(inode), 1, fd); //把inode 表写回文件

fseek(fd, (sizeof(block\_group)\*BLOCKS\_GROUP\_NUM + sizeof(bool)\*TOTAL\_INODES\_NUM

+ sizeof(inode)\*TOTAL\_INODES\_NUM + sizeof(bool)\*b\_bitmap\_index), SEEK\_SET);

fwrite(&block\_bitmap[b\_bitmap\_index], sizeof(bool), block\_num, fd); //把修改的数据块位图写回文件

fseek(fd, addr, SEEK\_SET);

fwrite(buffer, 1, count, fd); //把文件写入硬盘

fseek(fd, inode\_table[temp\_dir.inode].i\_block, SEEK\_SET);

fwrite(&temp\_dir, sizeof(dir), 1, fd); //把父目录写回文件

fclose(fd);

delete(buffer);

}

**check()**

/\*

\*功能:检测并回复文件系统

\*实现:检查每个数据块组的组描述符与该组数据块与inode的记录是否相符，

\* 检查超级块中的系统使用记录是否与实际相符(与数据块位图与i节点位图作对比)

\*/

void check()

{

unsigned int f\_b\_num = 0; //空闲数据块数目

unsigned int f\_i\_num = 0; //空闲inode数目

unsigned int total\_f\_b\_num=0;//总的空闲数据块数目

unsigned int total\_f\_i\_num=0;//总的空闲inode数目

int i,j; //循环控制变量

int start; //记录数据块位图及inode位图的起始位置

bool change = false;//标明是否有修改过

printf("检查文件系统中...\n");

for(i=0;i<BLOCKS\_GROUP\_NUM;i++)

{

f\_b\_num = 0;

f\_i\_num = 0;

start = i\*BLOCKS\_PER\_GROUP;

for(j=0;j<BLOCKS\_PER\_GROUP;j++)

{

if(block\_bitmap[start+j] == NOT\_USED) //统计每数据块组空闲数据块的数目

f\_b\_num++;

if(inode\_bitmap[start+j] == NOT\_USED) //统计每数据块组空闲inode的数目

f\_i\_num++;

}

if(\_block\_group[i].group\_desc.bg\_free\_blocks\_count != f\_b\_num) //不相等则修改组描述符

{

\_block\_group[i].group\_desc.bg\_free\_blocks\_count = f\_b\_num;

change = true;

}

if(\_block\_group[i].group\_desc.bg\_free\_inodes\_count != f\_i\_num) //不相等则修改组描述符

{

\_block\_group[i].group\_desc.bg\_free\_inodes\_count = f\_i\_num;

change = true;

}

total\_f\_b\_num += \_block\_group[i].group\_desc.bg\_free\_blocks\_count; //统计总的空闲数据块数

total\_f\_i\_num += \_block\_group[i].group\_desc.bg\_free\_inodes\_count; //统计总的空闲inode数

}

if(\_block\_group[0].super\_block.s\_free\_blocks\_count != total\_f\_b\_num) //不相等则修改超级块

{

for(i=0;i<BLOCKS\_PER\_GROUP;i++)

\_block\_group[0].super\_block.s\_free\_blocks\_count = total\_f\_b\_num;

change = true;

}

if(\_block\_group[0].super\_block.s\_free\_inodes\_count != total\_f\_i\_num) //不相等则修改超级块

{

for(i=0;i<BLOCKS\_PER\_GROUP;i++)

\_block\_group[0].super\_block.s\_free\_inodes\_count = total\_f\_i\_num;

change = true;

}

if(!change)

printf("系统没有异常\n");

else //将修改写回

{

printf("系统出现异常，正在修复中...\n");

fd = fopen("LINUX\_FILE\_SYSTEM","rb+");

if(fd == NULL)

{

printf("linux\_file\_system file create failed !\n");

system("pause");

}

fseek(fd,0,SEEK\_SET);

fwrite(&\_block\_group[0],sizeof(block\_group),BLOCKS\_GROUP\_NUM,fd); //把数据块组写到文件上

fwrite(&inode\_bitmap[0],sizeof(bool),TOTAL\_INODES\_NUM,fd); //把inode 位图写到文件上

fwrite(&inode\_table[0],sizeof(inode),TOTAL\_INODES\_NUM,fd); //把inode 表写到文件上

fwrite(&block\_bitmap[0],sizeof(bool),TOTAL\_BLOCKS\_NUM,fd); //把数据块位图写到文件上

printf("修复成功!\n");

}

}

**还有一些重要的辅助函数**

**getLastDir()**

/\*功能：解析出最终的那个路径("root/目录/第一个目录"通过此函数解析得到"第一个目录")

\*

\*path : 待解析的路径

\*temp\_dir ： 可理解为返回值，将解析到的目的路径赋值给temp\_dir

\*/

bool getLastDir(char \*path, dir &temp\_dir)

**findFullPath()**

/\*

\*功能:查找出完整的路径 (例如根据 "/第一个目录" 查找出 "root/目录/第一个目录 ")

\*

\*算法步骤:递归查找上一级目录，直到找到根目录为止

\*

\*temp\_dir : 需要被查找的目录

\*/

void findFullPath(dir temp\_dir)

**getDirName()**

/\*

\*功能：根据pos查找某一层级的目录名(例如"root/一层/二层" 若pos = 1,则找到root,若pos为2,则找到一层，类推...)

\*

\*算法：利用strtok函数进行字符串分割

\*

\*path : 待分解的路径

\*pos : 分解级数

\*dirName : 可理解为返回值，将查找到的结果赋值给dirName

\*/

bool getDirName(char \*path, int pos, char \*dirName)

**findFinalDir()**

/\*

\*功能：不断查找路径中给定的pos级目录并验证其正确性，直到查找到最后一级,即最终目录

\*

\*path : 待解析路径

\*pos : 标志当前处于第几级目录

\*dirName : 查找到pos级别的目录名赋值给dirName

\*temp\_dir ： 查找到的路径中的最后一级目录赋值给temp\_dir

\*/

bool findFinalDir(char \*path, int &pos, char \*dirName, dir &temp\_dir)

**dividePathAndName()**

/\*

\*功能：分离出路径与目录(例如 "root/a/b/c" 分离出"root>a>b" 和 "c" ,c是待创建的目录或文件)

\*

\*实现：利用strrchr函数

\*

\*path : 待分解的路径

\*dirName : 分解得到的名字结果赋值给dirName

\*temp\_dir：分解得到的目录结果赋值给temp\_dir

\*/

bool dividePathAndName(char \*path,char \*dirName, dir &temp\_dir)

**getBlock()**

/\*

\*功能: 分配数据块空间

\*

\*实现: 找到一段连续的空间能够容纳所申请的长度

\*

\*len : 所申请的数据空间的长度(盘块数目)

\*b\_index : 分配的数据块在数据块位图的起始位置(返回时赋值)

\* return : 实际分配的连续区域的起始地址(物理地址)

\*/

long getBlock(int len, int \*b\_index)

**freeBlock()**

/\*

\*功能:根据给出的起始地址加上数据块数目来释放这部分数据块

\*

\*len : 所用的数据块数目

\*pos : pos为要删除的数据块在数据块位图的起始位置

\*/

void freeBlock(int len, int pos)

**getInode()**

/\*

\*功能:分配inode节点

\*

\*实现:检查i节点位图是否有空闲的i节点可用，随之更新系统信息

\*

\*返回分配的i节点号

\*/

int getInode()

**freeInode()**

/\*

\*功能:释放inode节点

\*

\*算法步骤:根据要释放的inode号，更新超级块、组描述符、inode位图

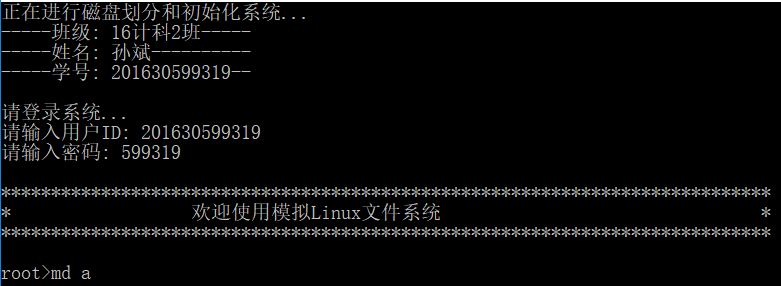
\*/

void freeInode(int inode)

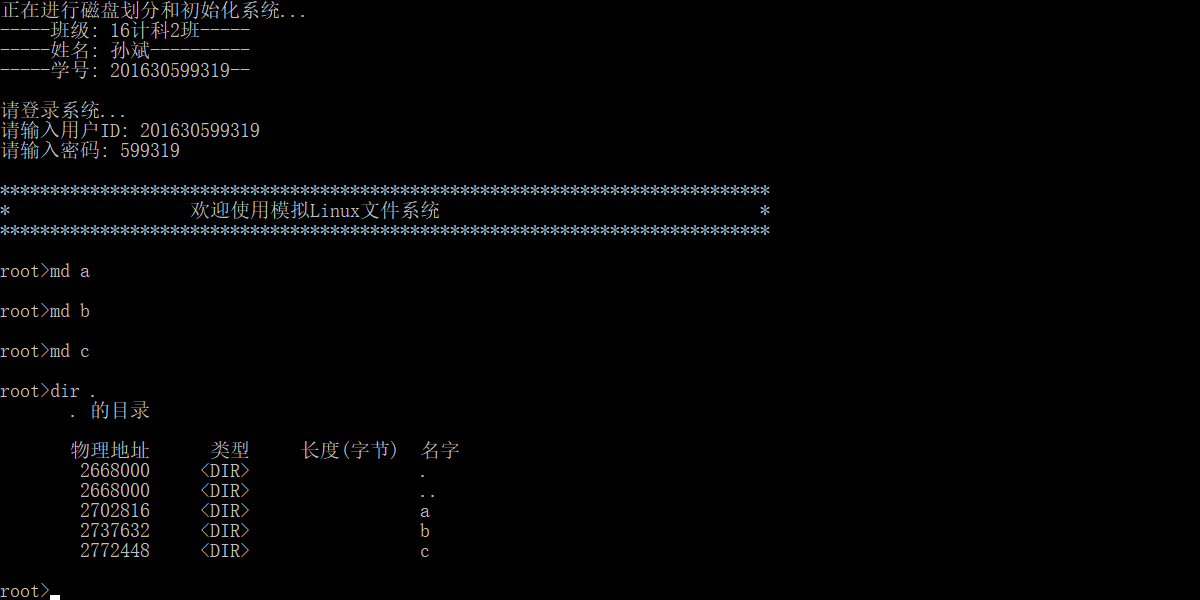
具体文件参考源代码文件

1. **结果分析**

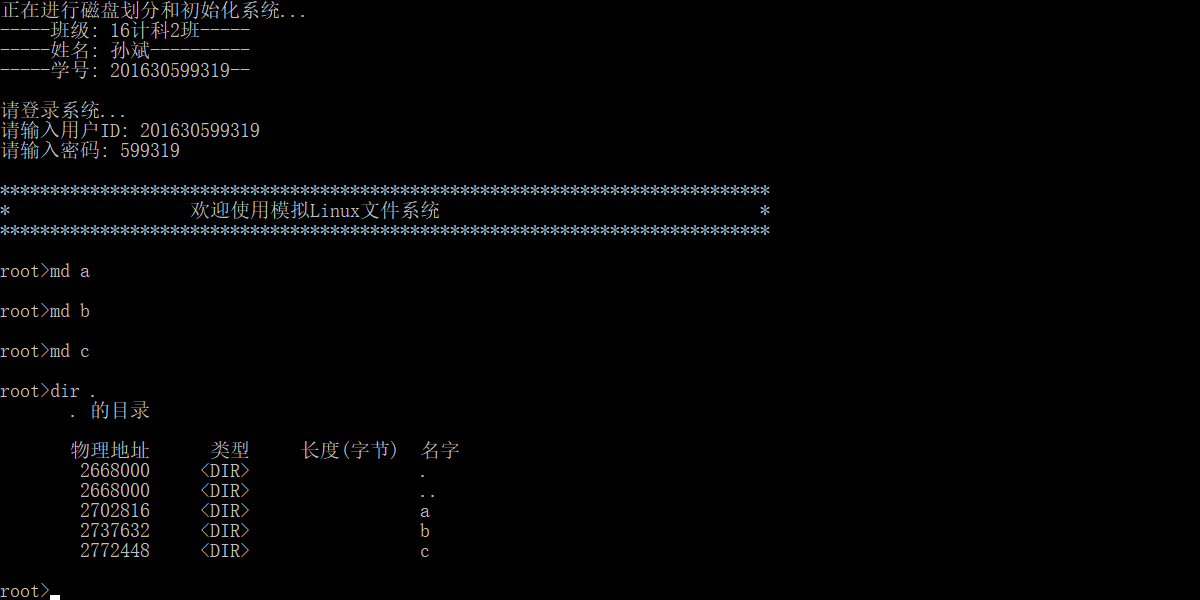
初始化系统及登录



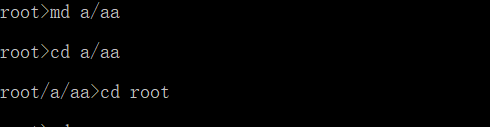
md创建目录



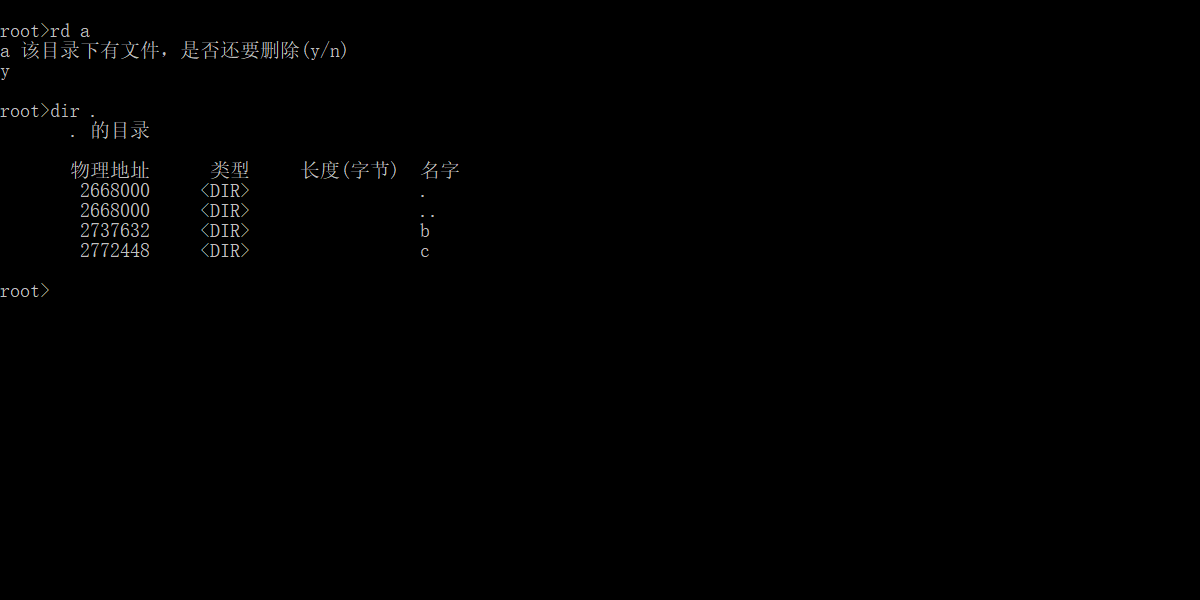
dir显示目录情况



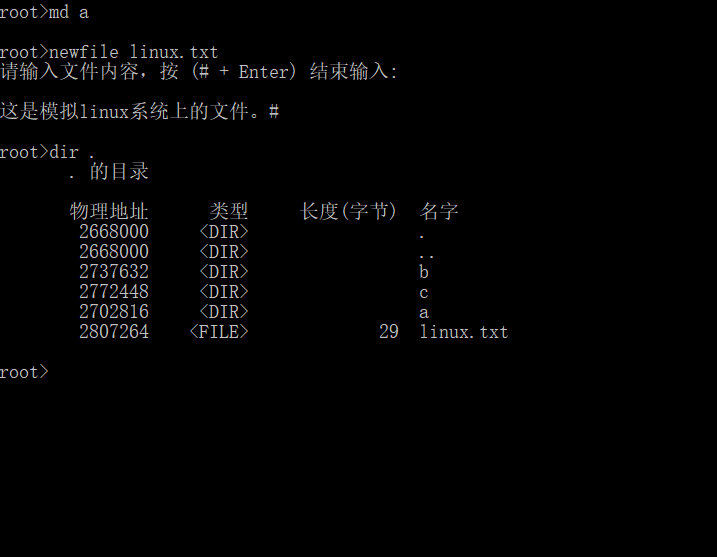
cd切换当前目录



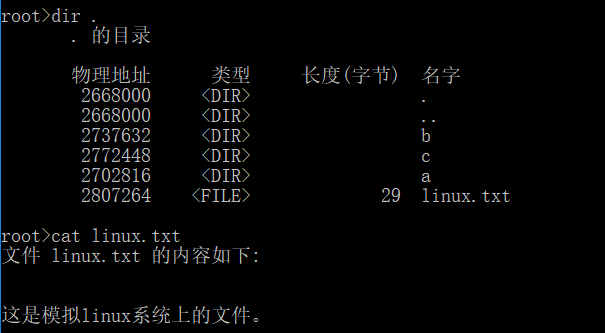
rd删除目录



newfile新建文件



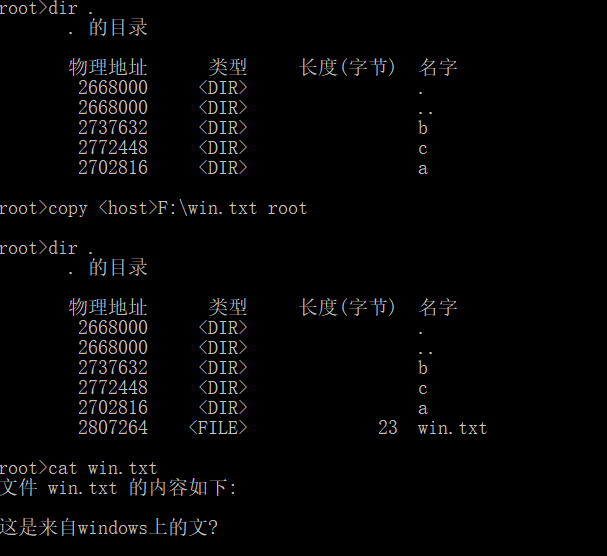
cat查看文件



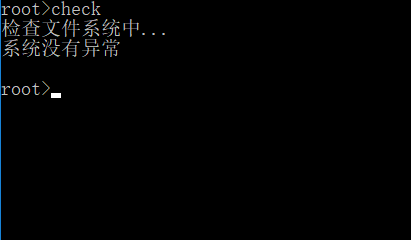
del删除文件



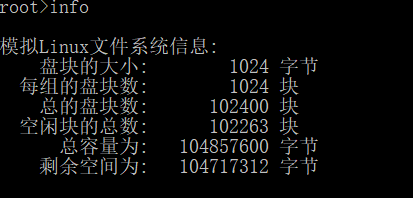
copy拷贝文件



check检查系统



info显示系统信息



程序测试无误

1. **课程设计总结**

这次课程设计刚开始看到题目的时候没什么很明确的想法，有点毫无头绪，无从下手，后面从网上找了很多linux文件系统的资料来看和理解，最关键是看了一篇讲ext2文件系统的文章(<http://docs.linuxtone.org/ebooks/C&CPP/c/ch29s02.html>)，对本系统数据结构的构造有很大帮助。

然后后面就是熟悉程序中内存与文件系统磁盘空间之间的数据交互过程，也对编程语言中的基本的那些数据格式整数和字符之类的是如何存储在硬盘上有了一个更加清晰直观的了解，对于C语言中的关于文件读取和字符串处理的函数也更加熟练，尤其是对字符串与字符数组的区别和字符串的拷贝也有了更加微观的理解。

同时也提升了自己的整体编码能力和完善了自己的代码风格，自己也拥有了更多的编程经验，收获很大。