**实验3文件系统**

#include <stdio.h>

#include "string.h"

#include "stdlib.h"

#include "time.h"

#include <sys/ioctl.h>

#include <termios.h>

#include <unistd.h> /\* for STDIN\_FILENO \*/

#define blocks 4611 // 1+1+1+512+4096,总块数

#define blocksiz 512 //每块字节数

#define inodesiz 64 //索引长度

#define data\_begin\_block 515 //数据开始块

#define dirsiz 32 //目录体长度

#define EXT2\_NAME\_LEN 15 //文件名长度

#define PATH "vdisk" //文件系统

typedef struct ext2\_group\_desc //组描述符 68 字节

{

char bg\_volume\_name[16]; //卷名

int bg\_block\_bitmap; //保存块位图的块号

int bg\_inode\_bitmap; //保存索引结点位图的块号

int bg\_inode\_table; //索引结点表的起始块号

int bg\_free\_blocks\_count; //本组空闲块的个数

int bg\_free\_inodes\_count; //本组空闲索引结点的个数

int bg\_used\_dirs\_count; //本组目录的个数

char psw[16]; //password

char bg\_pad[24]; //填充(0xff)

} ext2\_group\_desc;

typedef struct ext2\_inode //索引节点 64 字节

{

int i\_mode; //文件类型及访问权限 1:普通文件，2:目录

int i\_blocks; //文件内容占用的数据块个数

int i\_size; //大小(字节)

time\_t i\_atime; //访问时间

time\_t i\_ctime; //创建时间

time\_t i\_mtime; //修改时间

time\_t i\_dtime; //删除时间

int i\_block[8]; //指向数据块的指针

char i\_pad[24]; //填充 1(0xff)

} ext2\_inode;

typedef struct ext2\_dir\_entry //目录体 32 字节

{

int inode; //索引节点号

int rec\_len; //目录项长度

int name\_len; //文件名长度

int file\_type; //文件类型(1:普通文件，2:目录…)

char name[EXT2\_NAME\_LEN]; //文件名

char dir\_pad; //填充

} ext2\_dir\_entry;

/\*定义全局变量\*/

ext2\_group\_desc group\_desc; //组描述符

ext2\_inode inode;

ext2\_dir\_entry dir; //目录体（目录体，内容可能是文件也可能是目录）

FILE \*f; /\*文件指针\*/

unsigned int last\_allco\_inode = 0; //上次分配的索引节点号

unsigned int last\_allco\_block = 0; //上次分配的数据块号

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int getch() // 使用方法，在需要不显示输入的是什么的地方调用，返回值为用户输入的字符。

{

int ch;

struct termios oldt, newt;

tcgetattr(STDIN\_FILENO, &oldt); // 用来获取终端参数，成功返回零；失败返回非零

newt = oldt;

newt.c\_lflag &= ~(ECHO | ICANON);

tcsetattr(STDIN\_FILENO, TCSANOW, &newt);

ch = getchar();

tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);

return ch;

}

/\*\*\*\*\*\*\*\*格式化文件系统\*\*\*\*\*\*\*\*\*\*/

/\*

\* 初始化组描述符

\* 初始化数据块位图

\* 初始化索引节点位图

\* 初始化索引节点表 -添加一个索引节点

\* 第一个数据块中写入当前目录和上一目录

\*/

int initialize(ext2\_inode \*cu);

int format(ext2\_inode \*current)

{

FILE \*fp = NULL;

int i;

unsigned int zero[blocksiz / 4]; //零数组，用来初始化块为 0

time\_t now;

time(&now);

while (fp == NULL)

fp = fopen(PATH, "w+");

for (i = 0; i < blocksiz / 4; i++)

zero[i] = 0;

for (i = 0; i < blocks; i++) //初始化所有 4611 块为 0

{

fseek(fp, i \* blocksiz, SEEK\_SET);

fwrite(&zero, blocksiz, 1, fp);

}

//初始化组描述符

strcpy(group\_desc.bg\_volume\_name, "Volume\_name"); //初始化卷名为abcd

group\_desc.bg\_block\_bitmap = 1; //保存块位图的块号

group\_desc.bg\_inode\_bitmap = 2; //保存索引节点位图的块号

group\_desc.bg\_inode\_table = 3; //索引节点表的起始块号

group\_desc.bg\_free\_blocks\_count = 4095; //除去一个初始化目录。空闲数据块的个数

group\_desc.bg\_free\_inodes\_count = 4095;

group\_desc.bg\_used\_dirs\_count = 1;

strcpy(group\_desc.psw, "123");

fseek(fp, 0, SEEK\_SET);

fwrite(&group\_desc, sizeof(ext2\_group\_desc), 1, fp); //第一块为组描述符

//初始化数据块位图和索引节点位图，第一位置为 1

zero[0] = 0x80000000;

fseek(fp, 1 \* blocksiz, SEEK\_SET);

fwrite(&zero, blocksiz, 1, fp); //第二块为块位图，块位图的第一位为 1

fseek(fp, 2 \* blocksiz, SEEK\_SET);

fwrite(&zero, blocksiz, 1, fp); //第三块为索引位图，索引节点位图的第一位为 1

//初始化索引节点表，添加一个索引节点

inode.i\_mode = 2;

inode.i\_blocks = 1;

inode.i\_size = 64;

inode.i\_ctime = now;

inode.i\_atime = now;

inode.i\_mtime = now;

inode.i\_dtime = 0;

fseek(fp, 3 \* blocksiz, SEEK\_SET);

fwrite(&inode, sizeof(ext2\_inode), 1, fp); //第四块开始为索引节点表

//向第一个数据块写 当前目录

dir.inode = 0;

dir.rec\_len = 32; //默认目录体为32字节

dir.name\_len = 1;

dir.file\_type = 2;

strcpy(dir.name, "."); //当前目录

fseek(fp, data\_begin\_block \* blocksiz, SEEK\_SET);

fwrite(&dir, sizeof(ext2\_dir\_entry), 1, fp);

//当前目录之后写 上一目录

dir.inode = 0; //因为是根目录所以上一目录就是当前目录

dir.rec\_len = 32;

dir.name\_len = 2;

dir.file\_type = 2;

strcpy(dir.name, ".."); //上一目录

fseek(fp, data\_begin\_block \* blocksiz + dirsiz, SEEK\_SET);

fwrite(&dir, sizeof(ext2\_dir\_entry), 1, fp); //第data\_begin\_block+1 =516 块开始为数据

//current = &inode;

initialize(current); //将指针指向根目录

// last\_allco\_inode = 0; //上次分配的索引节点号

// last\_allco\_block = 0; //上次分配的数据块号

// current->i\_mode = 2;

// current->i\_blocks = 1;

// current->i\_size = 64;

// current->i\_ctime = now;

// current->i\_atime = now;

// current->i\_mtime = now;

// current->i\_dtime = 0;

printf("\n!!!!!!!!inode.i\_size:%d\n", inode.i\_size);

fclose(fp);

return 0;

}

//返回目录的起始存储位置，每个目录 32 字节

int dir\_entry\_position(int dir\_entry\_begin, int i\_block[8]) // dir\_entry\_begin目录体的相对开始字节

{

int dir\_blocks = dir\_entry\_begin / 512; // 存储目录需要的块数

int block\_offset = dir\_entry\_begin % 512; // 块内偏移字节数

int a;

FILE \*fp = NULL;

if (dir\_blocks <= 5) //前六个直接索引

return data\_begin\_block \* blocksiz + i\_block[dir\_blocks] \* blocksiz + block\_offset;

else //间接索引

{

while (fp == NULL)

fp = fopen(PATH, "r+");

dir\_blocks = dir\_blocks - 6;

if (dir\_blocks < 128) //一个块 512 字节，一个int为 4 个字节 一级索引有 512/4= 128 个

{

int a;

fseek(fp, data\_begin\_block \* blocksiz + i\_block[6] \* blocksiz + dir\_blocks \* 4, SEEK\_SET);

fread(&a, sizeof(int), 1, fp);

return data\_begin\_block \* blocksiz + a \* blocksiz + block\_offset;

}

else //二级索引

{

dir\_blocks = dir\_blocks - 128;

fseek(fp, data\_begin\_block \* blocksiz + i\_block[7] \* blocksiz + dir\_blocks / 128 \* 4, SEEK\_SET);

fread(&a, sizeof(int), 1, fp);

fseek(fp, data\_begin\_block \* blocksiz + a \* blocksiz + dir\_blocks % 128 \* 4, SEEK\_SET);

fread(&a, sizeof(int), 1, fp);

return data\_begin\_block \* blocksiz + a \* blocksiz + block\_offset;

}

fclose(fp);

}

}

/\*在当前目录 打开一个目录

current指向新打开的当前目录（ext2\_inode）

\*/

int Open(ext2\_inode \*current, char \*name)

{

FILE \*fp = NULL;

int i;

while (fp == NULL)

fp = fopen(PATH, "r+");

for (i = 0; i < (current->i\_size / 32); i++)

{

fseek(fp, dir\_entry\_position(i \* 32, current->i\_block), SEEK\_SET); //定位目录的偏移位置

fread(&dir, sizeof(ext2\_dir\_entry), 1, fp);

if (!strcmp(dir.name, name))

{

if (dir.file\_type == 2) //目录

{

fseek(fp, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET);

fread(current, sizeof(ext2\_inode), 1, fp);

fclose(fp);

return 0;

}

}

}

fclose(fp);

return 1;

}

/\*\*\*\*\*\*\*\*关闭当前目录\*\*\*\*\*\*\*\*\*\*/

/\*

关闭时仅修改最后访问时间

返回时 打开上一目录 作为当前目录

\*/

int Close(ext2\_inode \*current)

{

time\_t now;

ext2\_dir\_entry bentry;

FILE \*fout;

fout = fopen(PATH, "r+");

time(&now);

current->i\_atime = now; //修改最后访问时间

fseek(fout, (data\_begin\_block + current->i\_block[0]) \* blocksiz, SEEK\_SET);

fread(&bentry, sizeof(ext2\_dir\_entry), 1, fout); // current's dir\_entry

fseek(fout, 3 \* blocksiz + (bentry.inode) \* sizeof(ext2\_inode), SEEK\_SET);

fwrite(current, sizeof(ext2\_inode), 1, fout); //写入文件系统中

fclose(fout);

return Open(current, "..");

}

/\*

read file content from directory 'current' in file 'name'

\*/

int Read(ext2\_inode \*current, char \*name)

{

FILE \*fp = NULL;

int i;

while (fp == NULL)

fp = fopen(PATH, "r+");

for (i = 0; i < (current->i\_size / 32); i++) //遍历当前目录的目录项，ext2\_inode \*current 指向当前目录，每个目录项32字节

{

fseek(fp, dir\_entry\_position(i \* 32, current->i\_block), SEEK\_SET); // 返回目录项的起始存储位置

fread(&dir, sizeof(ext2\_dir\_entry), 1, fp);

if (!strcmp(dir.name, name)) // 比较文件名是否相同

{

if (dir.file\_type == 1) // 如果文件名相同，且文件类型是文件的话（文件类型有目录和文件两种）

{

time\_t now;

ext2\_inode node;

char content\_char;

fseek(fp, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET); //根据目录体中保存的索引节点号，找到文件的inode位置

fread(&node, sizeof(ext2\_inode), 1, fp); // original inode，node为文件的inode信息

i = 0;

for (i = 0; i < node.i\_size; i++) // 读出大小为i\_size的文件，一次读一个char

{

fseek(fp, dir\_entry\_position(i, node.i\_block), SEEK\_SET); //根据指向数据块的文件指针i\_block，将默认的读写指针移动到文件的数据块中

fread(&content\_char, sizeof(char), 1, fp);

if (content\_char == 0xD) //0xD (ascii——回车\n)

printf("\n");

else

printf("%c", content\_char);

}

printf("\n");

time(&now);

node.i\_atime = now; // 修改访问时间

fseek(fp, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET);

fwrite(&node, sizeof(ext2\_inode), 1, fp); // update inode 将修改写入文件系统中

fclose(fp);

return 0;

}

}

}

fclose(fp);

return 1;

}

//寻找空索引

int FindInode()

{

FILE \*fp = NULL;

unsigned int zero[blocksiz / 4];

int i;

while (fp == NULL)

fp = fopen(PATH, "r+");

fseek(fp, 2 \* blocksiz, SEEK\_SET); // inode 位图

fread(zero, blocksiz, 1, fp); // zero保存索引节点位图

// unsigned int zero[128] ,每个int4字节，共128个，故一共能表示128\*4\*32位，=512\*8 没问题！

for (i = last\_allco\_inode; i < (last\_allco\_inode + blocksiz / 4); i++) //一个inode号是int存储，故为4字节，

// last\_allco\_inode + blocksiz / 4 其实i的绝对数值已经超出索引节点位图的存储范围，但是因为要判断last\_allco\_inode之前的索引节点有无空闲，所以后面计算的时候%取余即可。

{

if (zero[i % (blocksiz / 4)] != 0xffffffff) //当还有空闲的索引节点时； 一个int4字节，4\*8=32位

//i % (blocksiz / 4)是某个索引节点号， zero[i % (blocksiz / 4)]表示inode位图中的某段32位区域 （128\*4\*32）一共有128个这样的区域

{

unsigned int j = 0x80000000, k = zero[i % (blocksiz / 4)], l = i;

for (i = 0; i < 32; i++)

{

if (!(k & j)) // & 按位与，再取非，如果结果不为0，说明第i位有空闲，否则，j = j / 2， 考察下一位是否空闲

{

zero[l % (blocksiz / 4)] = zero[l % (blocksiz / 4)] | j; //如果空闲，将此位置1

group\_desc.bg\_free\_inodes\_count -= 1; //索引节点数减 1

fseek(fp, 0, 0); //移动到起始位置——组描述符所在块

fwrite(&group\_desc, sizeof(ext2\_group\_desc), 1, fp); //更新组描述符（索引节点数目信息）

fseek(fp, 2 \* blocksiz, SEEK\_SET);

fwrite(zero, blocksiz, 1, fp); //更新inode位图，zero存储的是整个inode位图，所以直接更新即可

last\_allco\_inode = l % (blocksiz / 4);

fclose(fp);

return l % (blocksiz / 4) \* 32 + i; // 返回空闲的inode号

}

else

j = j / 2; // 考察下一位

}

}

}

fclose(fp);

return -1;

}

//寻找空block

int FindBlock()

{

FILE \*fp = NULL;

unsigned int zero[blocksiz / 4];

int i;

while (fp == NULL)

fp = fopen(PATH, "r+");

fseek(fp, 1 \* blocksiz, SEEK\_SET);

fread(zero, blocksiz, 1, fp); // zero保存块位图

for (i = last\_allco\_block; i < (last\_allco\_block + blocksiz / 4); i++)

{

if (zero[i % (blocksiz / 4)] != 0xffffffff)

{

unsigned int j = 0X80000000, k = zero[i % (blocksiz / 4)], l = i;

for (i = 0; i < 32; i++)

{

if (!(k & j))

{

zero[l % (blocksiz / 4)] = zero[l % (blocksiz / 4)] | j;

group\_desc.bg\_free\_blocks\_count -= 1; //块数减 1

fseek(fp, 0, 0);

fwrite(&group\_desc, sizeof(ext2\_group\_desc), 1, fp);

fseek(fp, 1 \* blocksiz, SEEK\_SET);

fwrite(zero, blocksiz, 1, fp);

last\_allco\_block = l % (blocksiz / 4);

fclose(fp);

return l % (blocksiz / 4) \* 32 + i;

}

else

j = j / 2;

}

}

}

fclose(fp);

return -1;

}

//删除inode，更新inode节点位图

void DelInode(int len) //len是inode号，是一个unsigned int值

{

unsigned int zero[blocksiz / 4], i;

int j;

f = fopen(PATH, "r+");

fseek(f, 2 \* blocksiz, SEEK\_SET);

fread(zero, blocksiz, 1, f);

i = 0x80000000;

for (j = 0; j < len % 32; j++)

i = i / 2;

zero[len / 32] = zero[len / 32] ^ i;

fseek(f, 2 \* blocksiz, SEEK\_SET);

fwrite(zero, blocksiz, 1, f);

fclose(f);

}

//删除block块，更新块位图

void DelBlock(int len)

{

unsigned int zero[blocksiz / 4], i;

int j;

f = fopen(PATH, "r+");

fseek(f, 1 \* blocksiz, SEEK\_SET);

fread(zero, blocksiz, 1, f);

i = 0x80000000;

for (j = 0; j < len % 32; j++)

i = i / 2;

zero[len / 32] = zero[len / 32] ^ i;

fseek(f, 1 \* blocksiz, SEEK\_SET);

fwrite(zero, blocksiz, 1, f);

fclose(f);

}

void add\_block(ext2\_inode \*current, int i, int j) // 空间不够，故增加一个数据块来存放内容

{

FILE \*fp = NULL;

while (fp == NULL)

fp = fopen(PATH, "r+");

if (i < 6) //直接索引

{

current->i\_block[i] = j;

}

else

{

i = i - 6;

if (i == 0)

{

current->i\_block[6] = FindBlock();

fseek(fp, data\_begin\_block \* blocksiz + current->i\_block[6] \* blocksiz, SEEK\_SET);

fwrite(&j, sizeof(int), 1, fp);

}

else if (i < 128) //一级索引

{

fseek(fp, data\_begin\_block \* blocksiz + current->i\_block[6] \* blocksiz + i \* 4, SEEK\_SET);

fwrite(&j, sizeof(int), 1, fp);

}

else //二级索引

{

i = i - 128;

if (i == 0)

{

current->i\_block[7] = FindBlock();

fseek(fp, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz, SEEK\_SET);

i = FindBlock();

fwrite(&i, sizeof(int), 1, fp);

fseek(fp, data\_begin\_block \* blocksiz + i \* blocksiz, SEEK\_SET);

fwrite(&j, sizeof(int), 1, fp);

}

if (i % 128 == 0)

{

fseek(fp, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz + i / 128 \* 4, SEEK\_SET);

i = FindBlock();

fwrite(&i, sizeof(int), 1, fp);

fseek(fp, data\_begin\_block \* blocksiz + i \* blocksiz, SEEK\_SET);

fwrite(&j, sizeof(int), 1, fp);

}

else

{

fseek(fp, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz + i / 128 \* 4, SEEK\_SET);

fread(&i, sizeof(int), 1, fp);

fseek(fp, data\_begin\_block \* blocksiz + i \* blocksiz + i % 128 \* 4, SEEK\_SET);

fwrite(&j, sizeof(int), 1, fp);

}

}

}

}

// 为当前目录寻找一个空目录体

int FindEntry(ext2\_inode \*current)

{

FILE \*fout = NULL;

int location; //条目的绝对地址

int block\_location; //块号

int temp; //每个block 可以存放的INT 数量

int remain\_block; //剩余块数

location = data\_begin\_block \* blocksiz;

temp = blocksiz / sizeof(int);

fout = fopen(PATH, "r+");

if (current->i\_size % blocksiz == 0) //一个BLOCK 使用完后增加一个块

{

add\_block(current, current->i\_blocks, FindBlock());

current->i\_blocks++;

}

if (current->i\_blocks < 6) //前 6 个块直接索引

{

location += current->i\_block[current->i\_blocks - 1] \* blocksiz;

location += current->i\_size % blocksiz;

}

else if (current->i\_blocks < temp + 5) //一级索引

{

block\_location = current->i\_block[6];

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz + (current->i\_blocks - 6) \* sizeof(int), SEEK\_SET);

fread(&block\_location, sizeof(int), 1, fout);

location += block\_location \* blocksiz;

location += current->i\_size % blocksiz;

}

else //二级索引

{

block\_location = current->i\_block[7];

remain\_block = current->i\_blocks - 6 - temp;

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz + (int)((remain\_block - 1) / temp + 1) \* sizeof(int), SEEK\_SET);

fread(&block\_location, sizeof(int), 1, fout);

remain\_block = remain\_block % temp;

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz + remain\_block \* sizeof(int),

SEEK\_SET);

fread(&block\_location, sizeof(int), 1, fout);

location += block\_location \* blocksiz;

location += current->i\_size % blocksiz + dirsiz;

}

current->i\_size += dirsiz;

fclose(fout);

return location;

}

/\*\*\*\*\*\*\*\*\*创建文件或者目录\*\*\*\*\*\*\*\*\*/

/\*

\* type=1 创建文件

type=2 创建目录

\* current 当前目录索引节点

\* name 文件名或目录名

\*/

int Create(int type, ext2\_inode \*current, char \*name)

{

FILE \*fout = NULL;

int i;

int block\_location; // block location

int node\_location; // node location

int dir\_entry\_location; // dir entry location

time\_t now;

ext2\_inode ainode;

ext2\_dir\_entry aentry, bentry; // bentry保存当前系统的目录体信息

time(&now);

fout = fopen(PATH, "r+");

node\_location = FindInode(); // 寻找空索引

// 检查是否存在重复文件或目录名称

for (i = 0; i < current->i\_size / dirsiz; i++)

{

fseek(fout, dir\_entry\_position(i \* sizeof(ext2\_dir\_entry), current->i\_block), SEEK\_SET);

fread(&aentry, sizeof(ext2\_dir\_entry), 1, fout);

if (aentry.file\_type == type && !strcmp(aentry.name, name))

return 1;

}

fseek(fout, (data\_begin\_block + current->i\_block[0]) \* blocksiz, SEEK\_SET);

fread(&bentry, sizeof(ext2\_dir\_entry), 1, fout); // current's dir\_entry

if (type == 1) //文件

{

ainode.i\_mode = 1;

ainode.i\_blocks = 0; //文件暂无内容

ainode.i\_size = 0; //初始文件大小为 0

ainode.i\_atime = now;

ainode.i\_ctime = now;

ainode.i\_mtime = now;

ainode.i\_dtime = 0;

for (i = 0; i < 8; i++)

{

ainode.i\_block[i] = 0;

}

for (i = 0; i < 24; i++)

{

ainode.i\_pad[i] = (char)(0xff);

}

}

else //目录

{

ainode.i\_mode = 2; //目录

ainode.i\_blocks = 1; //目录 当前和上一目录

ainode.i\_size = 64; //初始大小 32\*2=64 //一旦新建一个目录，该目录下就有"."和".."

ainode.i\_atime = now;

ainode.i\_ctime = now;

ainode.i\_mtime = now;

ainode.i\_dtime = 0;

block\_location = FindBlock();

ainode.i\_block[0] = block\_location;

for (i = 1; i < 8; i++)

{

ainode.i\_block[i] = 0;

}

for (i = 0; i < 24; i++)

{

ainode.i\_pad[i] = (char)(0xff);

}

//当前目录

aentry.inode = node\_location;

aentry.rec\_len = sizeof(ext2\_dir\_entry);

aentry.name\_len = 1;

aentry.file\_type = 2;

strcpy(aentry.name, ".");

printf(".dir created.\n");

aentry.dir\_pad = 0;

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz, SEEK\_SET);

fwrite(&aentry, sizeof(ext2\_dir\_entry), 1, fout);

//上一级目录

aentry.inode = bentry.inode;

aentry.rec\_len = sizeof(ext2\_dir\_entry);

aentry.name\_len = 2;

aentry.file\_type = 2;

strcpy(aentry.name, "..");

aentry.dir\_pad = 0;

fwrite(&aentry, sizeof(ext2\_dir\_entry), 1, fout);

printf("..dir created.\n");

//一个空条目

aentry.inode = 0;

aentry.rec\_len = sizeof(ext2\_dir\_entry);

aentry.name\_len = 0;

aentry.file\_type = 0;

aentry.name[EXT2\_NAME\_LEN] = 0;

aentry.dir\_pad = 0;

fwrite(&aentry, sizeof(ext2\_dir\_entry), 14, fout); //清空数据块

} // end else

//保存新建inode

fseek(fout, 3 \* blocksiz + (node\_location) \* sizeof(ext2\_inode), SEEK\_SET);

fwrite(&ainode, sizeof(ext2\_inode), 1, fout);

// 将新建inode 的信息写入current 指向的数据块

aentry.inode = node\_location;

aentry.rec\_len = dirsiz;

aentry.name\_len = strlen(name);

if (type == 1)

{

aentry.file\_type = 1;

} //文件

else

{

aentry.file\_type = 2;

} //目录

strcpy(aentry.name, name);

aentry.dir\_pad = 0;

dir\_entry\_location = FindEntry(current);

fseek(fout, dir\_entry\_location, SEEK\_SET); //定位条目位置

fwrite(&aentry, sizeof(ext2\_dir\_entry), 1, fout);

//保存current 的信息,bentry 是current 指向的block 中的第一条

//ext2\_inode cinode;

fseek(fout, 3 \* blocksiz + (bentry.inode) \* sizeof(ext2\_inode), SEEK\_SET);

// fread(&cinode, sizeof(ext2\_inode), 1, fout);

// printf("after\_cinode.i\_size: %d\n", cinode.i\_size);

fwrite(current, sizeof(ext2\_inode), 1, fout);

fclose(fout);

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* write data to file 'name' in directory 'current'\* if there isn't this file in this directory ,remaind create a new one

\*/

int Write(ext2\_inode \*current, char \*name)

{

FILE \*fp = NULL;

ext2\_dir\_entry dir;

ext2\_inode node;

time\_t now;

char str;

int i;

while (fp == NULL)

fp = fopen(PATH, "r+");

while (1)

{

for (i = 0; i < (current->i\_size / 32); i++)

{

fseek(fp, dir\_entry\_position(i \* 32, current->i\_block), SEEK\_SET);

fread(&dir, sizeof(ext2\_dir\_entry), 1, fp);

if (!strcmp(dir.name, name))

{

if (dir.file\_type == 1)

{

fseek(fp, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET);

fread(&node, sizeof(ext2\_inode), 1, fp);

break;

}

}

}

if (i < current->i\_size / 32) // have file

break;

// Create(1,current,name); //have not file ,create a new file

printf("There isn't this file,please create it first\n");

return 0;

}

str = getch();

while (str != 27) // 没有检测到ESC（ascii = 27）之前一直读

{

printf("%c", str);

if (!(node.i\_size % 512)) // 需要增加数据块

{

add\_block(&node, node.i\_size / 512, FindBlock());

node.i\_blocks += 1;

}

fseek(fp, dir\_entry\_position(node.i\_size, node.i\_block), SEEK\_SET);

fwrite(&str, sizeof(char), 1, fp);

node.i\_size += sizeof(char);

if (str == 0x0d)

printf("%c", 0x0a);

str = getch();

if (str == 27)

break;

}

time(&now);

node.i\_mtime = now;

node.i\_atime = now;

fseek(fp, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET);

fwrite(&node, sizeof(ext2\_inode), 1, fp);

fclose(fp);

printf("\n");

return 0;

}

/\*\*\*\*\*\*\*\*\*\*ls命令\*\*\*\*\*\*\*\*/

/\*

\* 列出当前目录的文件和目录

\*/

void Ls(ext2\_inode \*current)

{

ext2\_dir\_entry dir;

int i, j;

char timestr[150];

ext2\_inode node;

f = fopen(PATH, "r+");

printf("Type\t\tFileName\tCreateTime\t\t\tLastAccessTime\t\t\tModifyTime\n");

printf("\n!!!!!!!!current->i\_size:%d\n", current->i\_size);

for (i = 0; i < current->i\_size / 32; i++)

{

fseek(f, dir\_entry\_position(i \* 32, current->i\_block), SEEK\_SET);

fread(&dir, sizeof(ext2\_dir\_entry), 1, f); // 读出目录项内容

fseek(f, 3 \* blocksiz + dir.inode \* sizeof(ext2\_inode), SEEK\_SET);

fread(&node, sizeof(ext2\_inode), 1, f); // 读出索引节点的内容

strcpy(timestr, "");

strcat(timestr, asctime(localtime(&node.i\_ctime)));

strcat(timestr, asctime(localtime(&node.i\_atime)));

strcat(timestr, asctime(localtime(&node.i\_mtime)));

for (j = 0; j < strlen(timestr) - 1; j++)

if (timestr[j] == '\n')

{

timestr[j] = '\t';

}

if (dir.file\_type == 1)

printf("File\t\t%s\t\t%s", dir.name, timestr);

else

printf("Directory\t%s\t\t%s", dir.name, timestr);

}

fclose(f);

}

int initialize(ext2\_inode \*cu)

{

f = fopen(PATH, "r+");

fseek(f, 3 \* blocksiz, 0);

fread(cu, sizeof(ext2\_inode), 1, f);

fclose(f);

return 0;

}

/\*\*\*\*\*\*\*\*\*修改文件系统密码\*\*\*\*\*\*\*\*\*/

/\*

\* 修改成功返回 0

\* 修改不成功返回 1

\*/

int Password()

{

char psw[16], ch[10];

printf("Please input the old password\n");

scanf("%s", psw);

if (strcmp(psw, group\_desc.psw) != 0)

{

printf("Password error!\n");

return 1;

}

while (1)

{

printf("Please input the new password:");

scanf("%s", psw);

while (1)

{

printf("Modify the password?[Y/N]");

scanf("%s", ch);

if (ch[0] == 'N' || ch[0] == 'n')

{

printf("You canceled the modify of your password\n");

return 1;

}

else if (ch[0] == 'Y' || ch[0] == 'y')

{

strcpy(group\_desc.psw, psw);

f = fopen(PATH, "r+");

fseek(f, 0, 0);

fwrite(&group\_desc, sizeof(ext2\_group\_desc), 1, f);

fclose(f);

return 0;

}

else

printf("Meaningless command\n");

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int login()

{

char psw[16];

printf("please input the password(init:123):");

scanf("%s", psw);

return strcmp(group\_desc.psw, psw);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void exitdisplay()

{

printf("Thank you for using~ Byebye!\n");

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*初始化文件系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*返回 1 初始化失败，返回 0 初始化成功\*/

int initfs(ext2\_inode \*cu)

{

f = fopen(PATH, "r+");

if (f == NULL)

{

// char ch[20];/\*\*\*\*\*\*\*\*\*\*\*/

char ch;

int i;

printf("File system couldn't be found. Do you want to create one?\n[Y/N]");

i = 1;

while (i)

{

scanf("%c", &ch); /\*\*\*\*\*\*\*/

switch (ch)

{

case 'Y':

case 'y': /\*\*\*\*\*\*\*\*/

if (format(cu) != 0)

return 1;

f = fopen(PATH, "r");

i = 0;

break;

case 'N':

case 'n': /\*\*\*\*\*\*\*/

exitdisplay();

return 1;

default:

printf("Sorry, meaningless command\n");

break;

}

}

}

fseek(f, 0, SEEK\_SET);

fread(&group\_desc, sizeof(ext2\_group\_desc), 1, f);

fseek(f, 3 \* blocksiz, SEEK\_SET);

fread(&inode, sizeof(ext2\_inode), 1, f);

fclose(f);

initialize(cu);

return 0;

}

/\*\*\*\*\*\*\*\*\*获取当前目录的目录名\*\*\*\*\*\*\*\*\*/

void getstring(char \*cs, ext2\_inode node)

{

ext2\_inode current = node;

int i, j;

ext2\_dir\_entry dir;

f = fopen(PATH, "r+");

Open(&current, ".."); // current指向上一目录

for (i = 0; i < node.i\_size / 32; i++)

{

fseek(f, dir\_entry\_position(i \* 32, node.i\_block), SEEK\_SET);

fread(&dir, sizeof(ext2\_dir\_entry), 1, f);

if (!strcmp(dir.name, "."))

{

j = dir.inode;

break;

}

}

for (i = 0; i < current.i\_size / 32; i++)

{

fseek(f, dir\_entry\_position(i \* 32, current.i\_block), SEEK\_SET);

fread(&dir, sizeof(ext2\_dir\_entry), 1, f);

if (dir.inode == j)

{

strcpy(cs, dir.name);

return;

}

}

}

/\*\*\*\*\*\*\*在当前目录删除目录或者文件\*\*\*\*\*\*\*\*\*\*\*/

int Delet(int type, ext2\_inode \*current, char \*name)

{

FILE \*fout = NULL;

int i, j, t, k, flag;

// int Nlocation,Elocation,Blocation,

int Blocation2, Blocation3;

int node\_location, dir\_entry\_location, block\_location,e\_location;

int block\_location2, block\_location3;

ext2\_inode cinode, tmpinode,einode;

ext2\_dir\_entry bentry, centry, dentry,eentry;

//一个空条目

dentry.inode = 0;

dentry.rec\_len = sizeof(ext2\_dir\_entry);

dentry.name\_len = 0;

dentry.file\_type = 0;

strcpy(dentry.name, "");

dentry.dir\_pad = 0;

fout = fopen(PATH, "r+");

t = (int)(current->i\_size / dirsiz); //总条目数

flag = 0; //是否找到文件或目录

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

{

dir\_entry\_location = dir\_entry\_position(i \* dirsiz, current->i\_block);

fseek(fout, dir\_entry\_location, SEEK\_SET);

fread(&centry, sizeof(ext2\_dir\_entry), 1, fout);

if ((strcmp(centry.name, name) == 0) && (centry.file\_type == type))

{

flag = 1;

j = i;

break;

}

}

if (flag)

{

node\_location = centry.inode; //inode号

fseek(fout, 3 \* blocksiz + node\_location \* sizeof(ext2\_inode), SEEK\_SET); //定位INODE 位置

fread(&cinode, sizeof(ext2\_inode), 1, fout);

block\_location = cinode.i\_block[0];

//删文件夹

if (type == 2)

{

while (cinode.i\_size > 2 \* dirsiz)

{

fseek(fout,dir\_entry\_position(cinode.i\_size-dirsiz,cinode.i\_block),0);

fread(&eentry, sizeof(ext2\_dir\_entry), 1, fout);

Delet(eentry.file\_type,&cinode,eentry.name);

}

{

DelBlock(block\_location);

DelInode(node\_location);

dir\_entry\_location = dir\_entry\_position(current->i\_size-dirsiz, current->i\_block); //找到current 指向条目的最后一条

fseek(fout, dir\_entry\_location, SEEK\_SET);

fread(&centry, dirsiz, 1, fout); //将最后一条条目存入centry

fseek(fout, dir\_entry\_location, SEEK\_SET);

fwrite(&dentry, dirsiz, 1, fout); //清空该位置

dir\_entry\_location -= data\_begin\_block \* blocksiz; //在数据中的相对位置

//如果这个位置刚好是一个块的起始位置，则删掉这个块（如果删除的文件夹刚好是给目录体多分配的一个数据块的第一个目录项，则应该释放这个块）

if (dir\_entry\_location % blocksiz == 0)

{

DelBlock((int)(dir\_entry\_location / blocksiz));

current->i\_blocks--; //所用数据块数-1

if (current->i\_blocks == 6) //如果只剩下6个直接索引，则删除那个一级子索引

DelBlock(current->i\_block[6]);

else if (current->i\_blocks == (blocksiz / sizeof(int) + 6))

{

int a;

fseek(fout, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz, SEEK\_SET);

fread(&a, sizeof(int), 1, fout);

DelBlock(a);

DelBlock(current->i\_block[7]);

}

else if (!((current->i\_blocks - 6 - blocksiz / sizeof(int)) % (blocksiz / sizeof(int))))

{

int a;

fseek(fout, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz + ((current->i\_blocks - 6 - blocksiz / sizeof(int)) / (blocksiz / sizeof(int))), SEEK\_SET);

fread(&a, sizeof(int), 1, fout);

DelBlock(a);

}

}

current->i\_size -= dirsiz;

if (j \* dirsiz < current->i\_size) //删除的条目如果不是最后一条，用centry覆盖（将最后一个条目覆盖到删除的条目中）

{

dir\_entry\_location = dir\_entry\_position(j \* dirsiz, current->i\_block);

fseek(fout, dir\_entry\_location, SEEK\_SET);

fwrite(&centry, dirsiz, 1, fout);

}

printf("The %s is deleted!", name);

}

}

//删文件

else

{

//删直接指向的块

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

{

if (cinode.i\_blocks == 0)

{

break;

}

block\_location = cinode.i\_block[i];

DelBlock(block\_location);

cinode.i\_blocks--;

}

//删一级索引中的块

if (cinode.i\_blocks > 0) //（即初始情况下：i\_blocks>=7）

{

block\_location = cinode.i\_block[6];

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz, SEEK\_SET);

for (i = 0; i < blocksiz / sizeof(int); i++)

{

if (cinode.i\_blocks == 0)

{

break;

}

fread(&Blocation2, sizeof(int), 1, fout);

DelBlock(Blocation2);

cinode.i\_blocks--;

}

DelBlock(block\_location); // 删除一级索引

}

if (cinode.i\_blocks > 0) //有二级索引存在

{

block\_location = cinode.i\_block[7];

for (i = 0; i < blocksiz / sizeof(int); i++)

{

fseek(fout, (data\_begin\_block + block\_location) \* blocksiz + i \* sizeof(int), SEEK\_SET);

fread(&Blocation2, sizeof(int), 1, fout);

fseek(fout, (data\_begin\_block + Blocation2) \* blocksiz, SEEK\_SET);

for (k = 0; i < blocksiz / sizeof(int); k++)

{

if (cinode.i\_blocks == 0)

{

break;

}

fread(&Blocation3, sizeof(int), 1, fout);

DelBlock(Blocation3);

cinode.i\_blocks--;

}

DelBlock(Blocation2); //删除二级索引

}

DelBlock(block\_location); // 删除一级索引

}

DelInode(node\_location); //删除文件的inode

dir\_entry\_location = dir\_entry\_position(current->i\_size-dirsiz, current->i\_block); //找到current（当前目录） 指向条目的最后一条

//dir\_entry\_position输入：相对位置，返回目录的起始存储位置

printf("num:%d\n", current->i\_size/dirsiz);

fseek(fout, dir\_entry\_location, SEEK\_SET); //这里读取的位置应该是第四个的末尾也就是五个长度-dirsiz

fread(&centry, dirsiz, 1, fout); //将最后一条条目存入centry

printf("last entryname: %s\n", centry.name);

printf("last entrynamelen: %d\n", centry.name\_len);

fseek(fout, dir\_entry\_location, SEEK\_SET);

fwrite(&dentry, dirsiz, 1, fout); //清空该位置

dir\_entry\_location -= data\_begin\_block \* blocksiz; //在数据中的位置

//如果这个位置刚好是一个块的起始位置，则删掉这个块

if (dir\_entry\_location % blocksiz == 0)

{

DelBlock((int)(dir\_entry\_location / blocksiz));

current->i\_blocks--;

if (current->i\_blocks == 6)

DelBlock(current->i\_block[6]);

else if (current->i\_blocks == (blocksiz / sizeof(int) + 6))

{

int a;

fseek(fout, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz, SEEK\_SET);

fread(&a, sizeof(int), 1, fout);

DelBlock(a);

DelBlock(current->i\_block[7]);

}

else if (!((current->i\_blocks - 6 - blocksiz / sizeof(int)) % (blocksiz / sizeof(int))))

{

int a;

fseek(fout, data\_begin\_block \* blocksiz + current->i\_block[7] \* blocksiz + ((current->i\_blocks - 6 - blocksiz / sizeof(int)) / (blocksiz / sizeof(int))), SEEK\_SET);

fread(&a, sizeof(int), 1, fout);

DelBlock(a);

}

}

current->i\_size -= dirsiz; //更新当前目录的大小

printf("num:%d\n", current->i\_size/dirsiz);

if (j \* dirsiz < current->i\_size) // 删除的条目如果不是最后一条，用centry 覆盖被删除的条目

{

dir\_entry\_location = dir\_entry\_position(j \* dirsiz, current->i\_block);

fseek(fout, dir\_entry\_location, SEEK\_SET);

fwrite(&centry, dirsiz, 1, fout);

}

}

fseek(fout, (data\_begin\_block + current->i\_block[0]) \* blocksiz, SEEK\_SET);

fread(&bentry, sizeof(ext2\_dir\_entry), 1, fout); // current's dir\_entry //读操作写操作之后读写指针都会变

// //

// printf("\ntest name:%s\n", bentry.name);

//fseek(fout, 3 \* blocksiz + (bentry.inode) \* sizeof(ext2\_inode), SEEK\_SET); //当前目录的inode

//fread(&cinode, sizeof(ext2\_inode), 1, fout);

//printf("cinode.i\_size: %d\n", cinode.i\_size);

// printf("current.i\_size: %d\n", current->i\_size);

// //

fseek(fout, 3 \* blocksiz + (bentry.inode) \* sizeof(ext2\_inode), SEEK\_SET); //当前目录的inode

fwrite(current, sizeof(ext2\_inode), 1, fout); //将current 修改的数据写回文件 //fwrite(&current, sizeof(ext2\_inode), 1, fout);

// // //

// printf("current.i\_size: %d\n", current->i\_size);

// fseek(fout, 3 \* blocksiz + (bentry.inode) \* sizeof(ext2\_inode), SEEK\_SET);

// fread(&tmpinode, sizeof(ext2\_inode), 1, fout);

// printf("tmpinode.i\_size: %d\n", tmpinode.i\_size);

}

else

{

fclose(fout);

return 1; //删除失败

}

fclose(fout);

return 0;//成功删除

}

void pwd (char \*str, ext2\_inode \*current){//显示当前目录的绝对路径

FILE \*fout = NULL;

char string[100];

char \*slash = "/";

int node\_location, dir\_entry\_location, block\_location;

ext2\_inode cinode;

ext2\_dir\_entry pentry, centry; //上级条目、当前条目

fout = fopen(PATH, "r+");

fseek(fout, (data\_begin\_block + current->i\_block[0]) \* blocksiz, SEEK\_SET); //定位到当前目录的.条目，即当前目录。

fread(&centry, sizeof(ext2\_dir\_entry), 1, fout); // current's dir\_entry //读操作写操作之后读写指针都会变

//printf("\ndentry name:%s\n", dentry.name);

fseek(fout, (data\_begin\_block + current->i\_block[0]) \* blocksiz + dirsiz, SEEK\_SET); //定位到当前目录的..条目，即上一级目录。

fread(&pentry, sizeof(ext2\_dir\_entry), 1, fout); // current's dir\_entry //读操作写操作之后读写指针都会变

//printf("\nbentry name:%s\n", bentry.name);

fseek(fout, 3 \* blocksiz + (pentry.inode) \* sizeof(ext2\_inode), SEEK\_SET); //上一级目录的inode

fread(&cinode, sizeof(ext2\_inode), 1, fout);

getstring(string, cinode);

//printf("上一个string为%s\n",string);

//printf("\n%s ", string);

//printf("\ni\_size:%d\n", cinode.i\_size);

current = &cinode;

if (centry.inode != pentry.inode){ //如果没有递归到根目录，继续递归

// if (string != "."){ //如果当前目录的上一级是根目录，那么路径不应该加入"."，默认home绝对路径是"/"

strcat(string, slash);

strcat(string, str);

strcpy(str, string);

//printf("上一个str为%s\n",str);

// }

pwd(str, current);

}

}

/\*main shell\*/

void shellloop(ext2\_inode currentdir)

{

char command[10], var1[10], var2[128], path[10];

ext2\_inode temp;

int i, j;

char currentstring[20];

char

ctable[14][10] = {"create", "delete", "cd", "close", "read", "write", "password", "format", "exit", "login", "logout", "ls", "pwd", "help"};

while (1)

{

getstring(currentstring, currentdir); //获取当前目录的目录名

printf("\n[%s]> ", currentstring);

scanf("%s", command);

for (i = 0; i < 14; i++)

if (!strcmp(command, ctable[i]))

break;

if (i == 0 || i == 1) //创建，删除 文件/目录 create f "" , delete f ""

{

scanf("%s", var1); /\*\*\*\*\*\*/

scanf("%s", var2);

if (var1[0] == 'f')

j = 1; //创建文件

else if (var1[0] == 'd')

j = 2; //创建目录

else

{

printf("the first variant must be [f/d]");

continue;

}

if (i == 0)

{

if (Create(j, &currentdir, var2) == 1)

printf("Failed! %s can't be created\n", var2);

else

printf("Congratulations! %s is created\n", var2);

}

else

{

if (Delet(j, &currentdir, var2) == 1)

printf("Failed! %s can't be deleted!\n", var2);

else

printf("Congratulations! %s is deleted!\n", var2);

}

}

else if (i == 2) // open == cd change dir

{

scanf("%s", var2);

i = 0;

j = 0;

temp = currentdir;

while (1)

{

path[i] = var2[j];

if (path[i] == '/')

{

if (j == 0)

initialize(&currentdir); // 将当前目录指针指向根目录/

else if (i == 0) // 目录名中不能包含/

{

printf("path input error!\n");

break;

}

else //遇到'/'，说明要进入一个目录（每次只进入一级目录(调用Open)，故后面要将暂存的目录地址尾部设置为'\0'）

{

path[i] = '\0';

if (Open(&currentdir, path) == 1)

{

printf("path input error!\n");

currentdir = temp;

}

}

i = 0; // 重新设置相对路径

}

else if (path[i] == '\0')

{

if (i == 0)

break;

if (Open(&currentdir, path) == 1)

{

printf("path input error!\n");

currentdir = temp;

}

break;

}

else

i++;

j++;

}

}

else if (i == 3) // close

{

/\*imagine the second variable suply number of layers to get out of\*/

scanf("%d", &i);

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

if (Close(&currentdir) == 1)

{

printf("Warning! the number %d is too large\n", i);

break;

}

}

else if (i == 4) // read

{

scanf("%s", var2);

if (Read(&currentdir, var2) == 1)

printf("Failed! The file can't be read\n");

}

else if (i == 5) // write

{

printf("没有检测到ESC(ascii = 27)之前一直写数据\n");

scanf("%s", var2);

if (Write(&currentdir, var2) == 1)

printf("Failed! The file can't be written\n");

}

else if (i == 6) // password

Password();

else if (i == 7) // format

{

while (1)

{

printf("Do you want to format the filesystem?\n It will be dangerous to your data.\n");

printf("[Y/N]");

scanf("%s", var1);

if (var1[0] == 'N' || var1[0] == 'n')

break;

else if (var1[0] == 'Y' || var1[0] == 'y')

{

format(&currentdir);

break;

}

else

printf("please input [Y/N]");

}

}

else if (i == 8) // exit

{

while (1)

{

printf("Do you want to exit from filesystem?[Y/N]\n");

scanf("%s", var2);

if (var2[0] == 'N' || var2[0] == 'n')

break;

else if (var2[0] == 'Y' || var2[0] == 'y')

return;

else

printf("\nplease input [Y/N]\n");

}

}

else if (i == 9) // login

printf("Failed! You havn't logged out yet\n");

else if (i == 10) // logout

{

while (i)

{

printf("Do you want to logout from filesystem?[Y/N]");

scanf("%s", var1);

if (var1[0] == 'N' || var1[0] == 'n')

break;

else if (var1[0] == 'Y' || var1[0] == 'y')

{

initialize(&currentdir);

while (1)

{

printf("command: ");

scanf("%s", var2);

if (strcmp(var2, "login") == 0)

{

if (login() == 0)

{

i = 0;

break;

}

}

else if (strcmp(var2, "exit") == 0)

return;

}

}

else

printf("please input [Y/N]");

}

}

else if (i == 11) // ls

Ls(&currentdir);

else if (i == 12) {

char string[100];

for(int j=0;j<100;j++)

string[j]=0;

getstring(currentstring, currentdir); //获取当前目录的目录名

// printf("\n%s\n ", currentstring);

pwd(string, &currentdir);

strcat(string, currentstring);

// printf("这个string为%s\n",string);

int i=0;

for( i=0;string[i+1];i++)

string[i]=string[i+1];

string[i]=0;

printf("%s\n",string);

}

else if (i == 13) {//help

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

printf(" \* An simulation of ext2 file system \*\n");

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

printf(" \* The available commands are: \*\n");

printf(" \* 01.change dir : cd+dir\_name 02.create dir : create d+dir\_name \*\n");

printf(" \* 03.create file : create f+file\_name 04.delete dir : delete d+dir\_name \*\n");

printf(" \* 05.delete file : delete f+file\_name 06.read file : read+file\_name \*\n");

printf(" \* 07.write file : write+file\_name 08.absolute path : pwd \*\n");

printf(" \* 09.close file : close+num 10.change pw : password \*\n");

printf(" \* 11.list items : ls 12.this menu : help \*\n");

printf(" \* 13.format disk : format 14.exit : exit \*\n");

printf(" \* 15.logout : logout \*\n");

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

}

else

printf("Failed! Command not available\n");

}

}

int main()

{

ext2\_inode cu; /\*current user\*/

printf("Hello! Welcome to Ext2\_like file system!\n");

if (initfs(&cu) == 1)

return 0;

if (login() != 0) /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

{

printf("Wrong password!It will terminate right away.\n");

exitdisplay();

return 0;

}

shellloop(cu);

exitdisplay();

return 0;

}