1. 实验代码

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

//#include <iostream.h>

using namespace std;

#define B 10 //存储块长度

#define L 500 //存储块总数

#define K 100 //保留区大小

#define BUSY 1

#define FREE 0

#define OK 1

#define ERROR 0

#define FILE\_BLOCK\_LENGTH (B-3) //文件分配磁盘块号数组长度

#define FILE\_NAME\_LENGTH (B-1) //最长文件名长度

#define FILE\_SIGN\_AREA ((L-1-K)/B+1) //保留区中文件标识符起始块号位图之后

#define FILE\_NUM FILE\_BLOCK\_LENGTH //目录内最多文件数目

#define BUFFER\_LENGTH 25 //打开文件表目中的缓冲区长度

#define INPUT\_LENGTH 100 //写文件时最大输入长度

#define OUTPUT\_LENGTH 100 //读文件时最大读出长度

struct filesign { //文件描述符

int file\_length; //文件长度

int filesign\_flag; //占用标识位

int file\_block; //文件分配磁盘块号数组实际长度

int file\_block\_ary[FILE\_BLOCK\_LENGTH]; //文件分配磁盘块号数组

};

struct contents { //目录项

char filename[FILE\_NAME\_LENGTH]; //文件名

int filesignnum; //文件描述符序号

};

struct openfilelist { //打开文件表表目

char buffer[BUFFER\_LENGTH]; //读写缓冲区

int pointer[2]; //读写指针文件内容的位置

int filesignnum; //文件描述符

int flag; //占用符

};

char ldisk[L][B]; //用字符数组模拟磁盘

openfilelist open\_list[FILE\_NUM]; //打开文件表

#define DIR 0

#define CREATE 1

#define DELETE 2

#define OPEN 3

#define CLOSE 4

#define READ 5

#define WRITE 6

#define HELP 7

#define LSEEK 8

#define EXIT 9

#define OPLIST 10

#define LDISK 11

#define CH\_LENGTH 20

//辅助函数

int show\_openlist(); //显示打开文件表,返回打开文件个数

void directory(); //显示目录文件详细信息

void show\_help(); //该文件系统的帮助

void show\_ldisk(); //显示磁盘内容（辅助用）

//核心函数

void read\_block(int, char \*); //文件系统与IO设备的接口函数，读取块

void write\_block(int, char \*); //文件系统与IO设备的接口函数，写入块

void Init(); //初始化文件系统

int create(char \*); //创建文件

int destroy(char \*); //删除文件

int open(char \*); //打开文件

int close(int); //关闭文件

int read(int, int, int); //读文件

int write(int, int, int); //写文件

int write\_buffer(int, int); //把缓冲区内容写入文件

int lseek(int, int); //定位文件指针

void Init\_block(char, int); //初始化字符数组块（辅助）

void read\_block(int i, char \*p)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*读磁盘块

该函数把逻辑块i的内容读入到指针p指向的内存位置

拷贝的字符个数为存储块的长度B。

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

{

char \* temp = (char \*)malloc(sizeof(char));

temp = p;

for (int a = 0; a < B;)

{

\*temp = ldisk[i][a];

a++;

temp++;

}

}

void write\_block(int i, char \*p)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*写磁盘块

该函数把指针p指向的内容写入逻辑块

拷贝的字符个数为存储块的长度B。

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

{

char \* temp = (char \*)malloc(sizeof(char));

temp = p;

for (int a = 0; a < B;)

{

ldisk[i][a] = \*temp;

a++;

temp++;

}

}

void Init\_block(char \*temp, int length)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*初始化一个字符数组块

处理的字符数组块长度为B

内容为0

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

{

int i;

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

{

temp[i] = '\0';

}

}

int write\_buffer(int index, int list)

{

int i;

int j;

int freed;

char temp[B];

int buffer\_length = BUFFER\_LENGTH;

for (i = 0; i < BUFFER\_LENGTH; i++)

{

if (open\_list[list].buffer[i] == '\0')

{

buffer\_length = i;

break;

}

}

int x = open\_list[list].pointer[0];

int y = open\_list[list].pointer[1];

int z = B - y; //当前块空闲容量

if (buffer\_length < z) //块容量可写入缓冲区不需要再分配

{

read\_block(x, temp);

strncat(temp + y, open\_list[list].buffer, buffer\_length); //缓冲区接入

write\_block(x, temp);

read\_block(index + FILE\_SIGN\_AREA, temp); //更改文件长

temp[1] += buffer\_length;

write\_block(index + FILE\_SIGN\_AREA, temp);

open\_list[list].pointer[0] = x;

open\_list[list].pointer[1] = y + buffer\_length; //更新文件读写指针

}

else //大于需分配新块

{

read\_block(index + FILE\_SIGN\_AREA, temp);

if (temp[2] + (buffer\_length - z) / B + 1 > FILE\_BLOCK\_LENGTH)

{

printf("文件分配数组不够分配\n");

return ERROR;

}

read\_block(x, temp);

strncat(temp + y, open\_list[list].buffer, z); //缓冲区接入的长度，填满当块

write\_block(x, temp);

for (i = 0; i < (buffer\_length - z) / B; i++)

{

for (j = K + FILE\_NUM; j < L; j++)

{

read\_block((j - K) / B, temp);

if (temp[(j - K) % B] == FREE)

{

freed = j;

break;

}

}

if (j == L)

{

printf("磁盘已满，分配失败\n");

return ERROR;

}

Init\_block(temp, B);

strncpy(temp, (open\_list[list].buffer + z + (i\*B)), B);

write\_block(freed, temp); //写入内容

read\_block((freed - K) / B, temp); //更改位图状态

temp[(freed - K) % B] = BUSY;

write\_block((freed - K) / B, temp); //写入磁盘，位图(文件内容所对应的)

read\_block(index + FILE\_SIGN\_AREA, temp);

temp[2] ++;

temp[2 + temp[2]] = freed;

write\_block(index + FILE\_SIGN\_AREA, temp);

}

for (j = K + FILE\_NUM; j < L; j++)

{

read\_block((j - K) / B, temp);

if (temp[(j - K) % B] == FREE)

{

freed = j;

break;

}

}

if (j == L)

{

printf("磁盘已满，分配失败\n");

return ERROR;

}

Init\_block(temp, B);

strncpy(temp, (open\_list[list].buffer + z + (i\*B)), (buffer\_length - z) % B);

write\_block(freed, temp); //写入内容

read\_block((freed - K) / B, temp); //更改位图状态

temp[(freed - K) % B] = BUSY;

write\_block((freed - K) / B, temp);

read\_block(index + FILE\_SIGN\_AREA, temp);

temp[2] ++;

temp[2 + temp[2]] = freed;

write\_block(index + FILE\_SIGN\_AREA, temp);

read\_block(index + FILE\_SIGN\_AREA, temp);

temp[1] += buffer\_length;

write\_block(index + FILE\_SIGN\_AREA, temp);

open\_list[list].pointer[0] = freed;

open\_list[list].pointer[1] = (buffer\_length - z) % B;

}

// printf("X = %d, Y = %d\n",open\_list[list].pointer[0],open\_list[list].pointer[1]);

}

int lseek(int index, int pos)

{

int i;

int list = -1;

char temp[B];

int pos\_i = pos / B;

int pos\_j = pos % B;

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == index)

{

list = i;

break;

}

}

if (list == -1)

{

printf("没找到当前索引号文件,操作失败..\n");

return ERROR;

}

if (open\_list[list].flag != BUSY)

{

printf("输入的索引号有误，操作失败...\n");

return ERROR;

}

read\_block(open\_list[list].filesignnum + FILE\_SIGN\_AREA, temp);

if (pos\_i > temp[2] - 1)

{

printf("异常越界,定位失败\n");

return ERROR;

}

open\_list[list].pointer[0] = temp[3 + pos\_i];

open\_list[list].pointer[1] = pos\_j;

return OK;

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

}

void Init()

{

int i;

char temp[B];

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

{

Init\_block(temp, B);

write\_block(i, temp);

}

for (i = K; i < L; i++)

{

read\_block((i - K) / B, temp);

temp[(i - K) % B] = FREE;

write\_block((i - K) % B, temp);

}

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

filesign temp\_cnt\_sign;

temp\_cnt\_sign.filesign\_flag = 1;

temp\_cnt\_sign.file\_length = 0;

temp\_cnt\_sign.file\_block = FILE\_BLOCK\_LENGTH;

Init\_block(temp, B);

temp[0] = temp\_cnt\_sign.filesign\_flag;

temp[1] = temp\_cnt\_sign.file\_length;

temp[2] = temp\_cnt\_sign.file\_block;

for (i = 0; i < FILE\_BLOCK\_LENGTH; i++)

{

temp[i + 3] = K + i;

}

write\_block(FILE\_SIGN\_AREA, temp);

//\*\*\*\*\*\*\*\*\*\*\*\*?

read\_block(0, temp);

for (i = 0; i < FILE\_NUM; i++)

{

temp[i] = FREE;

}

write\_block(0, temp);

}

int create(char filename[])

{

int i;

int frees;

int freed;

int freed2;

char temps[B];

char tempc[B];

char temp[B];

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

for (i = K; i < K + FILE\_NUM; i++)

{

read\_block((i - K) / B, temp);

if (temp[(i - K) % B] == BUSY)

{

read\_block(i, temp);

if (strncmp(temp + 1, filename, FILE\_NAME\_LENGTH) == 0)

{

printf("该目录已经存在文件名为%s的文件\n", filename);

return ERROR;

}

}

}

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

for (i = FILE\_SIGN\_AREA; i < K; i++)

{

read\_block(i, temp);

if (temp[0] == FREE)

{

frees = i;

break;

}

}

if (i == K)

{

printf("没有空闲的文件描述符\n");

return ERROR;

}

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

for (i = K; i < K + FILE\_NUM; i++)

{

read\_block((i - K) / B, temp);

if (temp[(i - K) % B] == FREE)

{

freed = i;

break;

}

}

if (i == K + FILE\_NUM)

{

printf("文件个数已达上限\n");

return ERROR;

}

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

for (i = K + FILE\_NUM; i < L; i++)

{

read\_block((i - K) / B, temp);

if (temp[(i - K) % B] == FREE)

{

freed2 = i;

break;

}

}

if (i == L)

{

printf("磁盘已满，分配失败\n");

return ERROR;

}

filesign temp\_filesign;

contents temp\_contents;

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

temp\_filesign.filesign\_flag = 1;

temp\_filesign.file\_length = 0;

temp\_filesign.file\_block = 1;

Init\_block(temps, B);

temps[0] = temp\_filesign.filesign\_flag;

temps[1] = temp\_filesign.file\_length;

temps[2] = temp\_filesign.file\_block;

temps[3] = freed2;

for (i = 4; i < FILE\_BLOCK\_LENGTH; i++)

{

temps[i] = '\0';

}

write\_block(frees, temps);

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

temp\_contents.filesignnum = frees - FILE\_SIGN\_AREA;

strncpy(temp\_contents.filename, filename, FILE\_NAME\_LENGTH);

Init\_block(tempc, B);

tempc[0] = temp\_contents.filesignnum;

tempc[1] = '\0';

strcat(tempc, temp\_contents.filename);

write\_block(freed, tempc);

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

read\_block((freed - K) / B, temp);

temp[(freed - K) % B] = BUSY;

write\_block((freed - K) / B, temp);

read\_block((freed2 - K) / B, temp);

temp[(freed2 - K) % B] = BUSY;

write\_block((freed2 - K) / B, temp);

read\_block(FILE\_SIGN\_AREA, temp);

temp[1]++;

write\_block(FILE\_SIGN\_AREA, temp);

return OK;

}

int destroy(char \* filename)

{

int i;

int dtys;

int dtyd;

int use\_block;

int index;

char temp[B];

char tempd[B];

for (i = K; i < K + FILE\_NUM; i++)

{

read\_block((i - K) / B, temp);

if (temp[(i - K) % B] == BUSY)

{

read\_block(i, temp);

if (strncmp(temp + 1, filename, FILE\_NAME\_LENGTH) == 0)

{

dtyd = i;

dtys = temp[0] + FILE\_SIGN\_AREA;

index = temp[0];

break;

}

}

}

if (i == K + FILE\_NUM)

{

printf("没有找到该文件\n");

return ERROR;

}

//\*\*

int list = -1;

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == index)

{

list = i;

break;

}

}

if (open\_list[list].flag == BUSY && list != -1)

{

printf("该文件已经被打开,需要关闭才能删除\n");

return ERROR;

}

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

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

read\_block(dtys, temp);

use\_block = temp[2];

for (i = 0; i < use\_block; i++)

{

read\_block((temp[i + 3] - K) / B, tempd);

tempd[(temp[i + 3] - K) % B] = FREE;

write\_block((temp[i + 3] - K) / B, tempd);

}

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

Init\_block(temp, B);

write\_block(dtys, temp);

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

Init\_block(temp, B);

write\_block(dtyd, temp);

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

read\_block((dtyd - K) / B, temp);

temp[(dtyd - K) % B] = FREE;

write\_block((dtyd - K) / B, temp);

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

read\_block(FILE\_SIGN\_AREA, temp);

temp[1]--;

write\_block(FILE\_SIGN\_AREA, temp);

return OK;

}

int open(char \* filename)

{

int i;

int opd;

int ops;

int list;

char temp[B];

int index;

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

for (i = K; i < K + FILE\_NUM; i++)

{

read\_block((i - K) / B, temp);

if (temp[(i - K) % B] == BUSY)

{

read\_block(i, temp);

if (strncmp(temp + 1, filename, FILE\_NAME\_LENGTH) == 0)

{

opd = i;

ops = temp[0];

// printf("opd: %d,ops: %d\n",opd,ops);

break;

}

}

}

if (i == K + FILE\_NUM)

{

printf("没有找到该文件\n");

return ERROR;

}

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

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == ops && open\_list[i].flag == BUSY)

{

printf("该文件已经被打开\n");

return ERROR;

}

}

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

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].flag != BUSY)

{

list = i;

break;

}

}

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

open\_list[list].filesignnum = ops;

open\_list[list].flag = BUSY;

index = open\_list[list].filesignnum;

lseek(index, 0);

Init\_block(open\_list[list].buffer, BUFFER\_LENGTH);

read\_block(open\_list[list].pointer[0], temp);

strncpy(open\_list[list].buffer, temp, BUFFER\_LENGTH);

return OK;

}

int close(int index)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*关闭文件

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

{

int i;

int list = -1;

char temp[B];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*根据index找表目

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == index)

{

list = i;

break;

}

}

if (list == -1) //没找到

{

printf("没找到当前索引号文件,操作失败...\n");

return ERROR;

}

if (open\_list[list].flag != BUSY) //输入的index对应文件没被打开

{

printf("输入的索引号有误，操作失败...\n");

return ERROR;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*将缓冲区的内容写入磁盘

/\* //测试用

for(i = 0 ; i < BUFFER\_LENGTH-1; i++ )

{

open\_list[list].buffer[i] = 5;

}

\*/

write\_buffer(index, list); //将当前list打开文件表对应的缓冲区入index索引号的文件

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*清楚操释放表目

Init\_block(open\_list[list].buffer, BUFFER\_LENGTH); //清除缓冲区

open\_list[list].filesignnum = FREE; //清除文件描述符

open\_list[list].flag = FREE; //清除占用标志位

open\_list[list].pointer[0] = NULL; //清楚指针

open\_list[list].pointer[1] = NULL;

return OK;

}

int read(int index, int mem\_area, int count)

/\*

从指定文件顺序读入count 个字

节mem\_area 指定的内存位置。读操作从文件的读写指针指示的位置

开始。

\*/

{

int i;

int list = -1;

char temp[B];

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*根据index找打开文件表表目

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == index)

{

list = i;

break;

}

}

if (list == -1) //没找到

{

printf("没找到当前索引号文件,操作失败...\n");

return ERROR;

}

if (open\_list[list].flag != BUSY) //输入的index对应文件没被打开

{

printf("输入的索引号有误，操作失败...\n");

return ERROR;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*根据index找文件描述符

char temp\_output[OUTPUT\_LENGTH];

Init\_block(temp\_output, OUTPUT\_LENGTH);

char output[OUTPUT\_LENGTH];

Init\_block(output, OUTPUT\_LENGTH);

read\_block(FILE\_SIGN\_AREA + index, temp);

int file\_length = temp[1]; //文件长度

int file\_block = temp[2]; //文件实际使用块

int file\_area;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*拷贝文件内容至temp\_output

for (i = 0; i < file\_block - 1; i++)

{

read\_block(FILE\_SIGN\_AREA + index, temp);

read\_block(temp[3 + i], temp);

strncpy(temp\_output + i \* B, temp, B);

}

read\_block(FILE\_SIGN\_AREA + index, temp);

read\_block(temp[3 + i], temp);

strncpy(temp\_output + i \* B, temp, B);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*当前文件读写坐标

int x = open\_list[list].pointer[0];

int y = open\_list[list].pointer[1];

for (i = 0; i < file\_block; i++)

{

read\_block(FILE\_SIGN\_AREA + index, temp);

if (temp[3 + i] == x)

{

break;

}

}

file\_area = i \* B + y; //转换文件内相对位置

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

{

output[i + mem\_area] = temp\_output[i + file\_area];

}

printf("%s\n", output + mem\_area);

return OK;

}

int write(int index, int mem\_area, int count)

/\*

把mem\_area 指定的内存位置开

始的ount 个字节顺序写入指定文件写操作从文件的读写指针指示

的位置开始。

\*/

{

int i;

int list = -1;

int input\_length;

char temp[B];

//\*\*\*\*\*\*\*\*\*\*\*\*\*根据index找到文件打开表

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].filesignnum == index)

{

list = i;

break;

}

}

if (list == -1) //没找到

{

printf("没找到当前索引号文件,操作失败...\n");

return ERROR;

}

if (open\_list[list].flag != BUSY) //输入的index对应文件没被打开

{

printf("输入的索引号有误，操作失败...\n");

return ERROR;

}

char input[INPUT\_LENGTH];

Init\_block(input, INPUT\_LENGTH);

i = 0;

fflush(stdin);

while (scanf("%c", &input[i]))

{

if (input[i] == '\n') //回车时终止读入

{

input[i] = '\0';

break;

}

i++;

}

input\_length = i;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*考虑输入串的长度需要写入的长度为input\_length - mem\_area

//缓冲区容量可写入输入内容不需要再分配

if (count <= BUFFER\_LENGTH)

{

strncat(open\_list[list].buffer, input + mem\_area, count); //存入缓冲区

}

//大于缓冲区长度分次写入，需要分块写入

else

{

int rest; //当前缓冲区空闲容量

for (i = 0; i < BUFFER\_LENGTH; i++)

{

if (open\_list[list].buffer[i] == FREE)

{

rest = BUFFER\_LENGTH - i;

break;

}

}

//第一部分，缓冲区有一定容量rest 将缓冲区写入文件，清空缓冲区

strncat(open\_list[list].buffer + BUFFER\_LENGTH - rest, input + mem\_area, rest);

write\_buffer(index, list);

Init\_block(open\_list[list].buffer, BUFFER\_LENGTH);

//第二部分，循环(input\_length - mem\_area)/BUFFER\_LENGTH 块缓冲区写入文件

for (i = 0; i < (count / BUFFER\_LENGTH) - 1; i++)

{

strncpy(open\_list[list].buffer, (input + mem\_area) + rest + i \* BUFFER\_LENGTH, BUFFER\_LENGTH);

write\_buffer(index, list);

Init\_block(open\_list[list].buffer, BUFFER\_LENGTH);

}

//第三部分，(count%BUFFER\_LENGTH)剩余未满一块的写入缓冲区

Init\_block(open\_list[list].buffer, BUFFER\_LENGTH);

strncpy(open\_list[list].buffer, (input + mem\_area) + rest + i \* BUFFER\_LENGTH, count%BUFFER\_LENGTH);

int buffer\_start;

}

return OK;

}

void directory()

//列表显示所有文件及其长度。

{

int i;

int filenum;

int filelength;

char filename[FILE\_NAME\_LENGTH];

char temp[B];

char tempd[B];

char temps[B];

read\_block(FILE\_SIGN\_AREA, temp);

filenum = temp[1]; //实际存在的文件个数

printf("\n");

if (filenum == 0)

{

printf("\t\t\t\t该目录下没有文件\n");

}

for (i = 0; i < FILE\_NUM; i++)

{

read\_block(temp[3 + i], tempd); //读取目录项

if (tempd[0] != 0)

{

read\_block(tempd[0] + FILE\_SIGN\_AREA, temps); //读取文件描述符

if (temps[0] == BUSY && tempd[0] != 0)

{

filelength = temps[1];

strcpy(filename, tempd + 1);

printf("%-10s\t\t%-2d字节\n", filename, filelength);

}

}

}

if (filenum != 0)

{

printf("\t\t\t\t共%d个文件\n", filenum);

}

}

int show\_openlist()

{

int i, j;

int openfile = 0;

char temp[B];

int index;

printf("\n索引号\t\t大小\t\t文件名\n");

for (i = 0; i < FILE\_NUM; i++)

{

if (open\_list[i].flag == BUSY)

{

index = open\_list[i].filesignnum;

printf(" %-2d", index);

openfile++;

read\_block(FILE\_SIGN\_AREA + index, temp);

printf("\t\t %-2d", temp[1]);

//根据Yndex文件描述符序号找到其目录项

for (j = K; j < K + FILE\_NUM; j++)

{

read\_block(j, temp);

if (temp[0] == index)

{

printf("\t\t%-10s\n", temp + 1);

}

}

}

}

return openfile;

}

void show\_help()

{

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

printf("\*\* 命令 说明 \*\*\n");

printf("\*\* dir 显示目录内容 \*\*\n");

printf("\*\* create+filename 新建以filename为文件名的文件 \*\*\n");

printf("\*\* delete+filename 删除以filename为文件名的文件 \*\*\n");

printf("\*\* open+filename 打开以filename为文件名的文件 \*\*\n");

printf("\*\* close 关闭index为索引号的文件 \*\*\n");

printf("\*\* read 请根据提示，接着要求输入索引号以及读取长度进行读\*\*\n");

printf("\*\* 取文件操作 \*\*\n");

printf("\*\* write 请根据提示，接着要求输入索引号以及写入长度进行写\*\*\n");

printf("\*\* 入文件操作 \*\*\n");

printf("\*\* lseek 请根据提示，接着要求输入一个不大于文件长度的数字\*\*\n");

printf("\*\* 用于定位读写指针 \*\*\n");

printf("\*\* help 帮助 \*\*\n");

printf("\*\* exit 退出文件系统 \*\*\n");

}

void show\_ldisk()

{

int a, b;

for (a = 0; a < K + 30; a++)

{

printf("%-3d :", a);

for (b = 0; b < B; b++)

{

printf("%-3d ", ldisk[a][b]);

}

printf("\n");

}

}

void main()

{

printf("\t\t\t欢迎使用该文件系统 使用帮助如下\n");

show\_help();

Init();

create((char\*)"file1");

create((char\*)"file2");

create((char\*)"file3");

open((char\*)"file1");

open((char\*)"file2");

char ch[CH\_LENGTH];

Init\_block(ch, CH\_LENGTH);

while (gets\_s(ch))

{

int cmd;

char filename[FILE\_NAME\_LENGTH];

//初始化

cmd = -1;

Init\_block(filename, FILE\_NAME\_LENGTH);

if (strncmp("dir", ch, 3) == 0) //浏览目录dir(non)

{

cmd = DIR;

}

if (strncmp("create", ch, 6) == 0) //创建文件命令create(filename)

{

cmd = CREATE;

strcat(filename, ch + 7);

}

if (strncmp("delete", ch, 6) == 0) //删除文件命令delete(filename)

{

cmd = DELETE;

strcat(filename, ch + 7);

}

if (strncmp("open", ch, 4) == 0) //打开文件命令open(filename)

{

cmd = OPEN;

strcat(filename, ch + 5);

}

if (strncmp("close", ch, 5) == 0) //关闭文件命令close(index)

{

cmd = CLOSE;

}

if (strncmp("read", ch, 4) == 0) //读文件命令read(index)

{

cmd = READ;

}

if (strncmp("write", ch, 5) == 0) //写文件命令write(index)

{

cmd = WRITE;

}

if (strncmp("lseek", ch, 5) == 0) //指针命令lseek(index,pos)

{

cmd = LSEEK;

}

if (strncmp("oplist", ch, 6) == 0) // 查看打开文件表

{

cmd = OPLIST;

}

if (strncmp("exit", ch, 4) == 0) // 退出命令exit

{

cmd = EXIT;

break;

}

if (strncmp("ldisk", ch, 5) == 0) //查看硬盘内容（辅助用）

{

cmd = LDISK;

}

if (strncmp("help", ch, 4) == 0) //帮助命令help(non)

{

cmd = HELP;

}

int index, count, pos;

switch (cmd)

{

case DIR:

directory();

printf("----------------------------------------------\n");

break;

case CREATE:

if (create(filename) == OK)

printf("创建文件成功|\n");

printf("----------------------------------------------\n");

break;

case DELETE:

if (destroy(filename) == OK)

printf("删除文件成功|\n");

printf("----------------------------------------------\n");

break;

case OPEN:

if (open(filename) == OK)

printf("打开文件成功|\n");

printf("----------------------------------------------\n");

break;

case CLOSE:

if (show\_openlist() == 0)

{

printf("当前没有文件被打开\n");

printf("----------------------------------------------\n");

break;

}

printf("请输入要关闭文件的索引号：\n");

scanf("%d", &index);

if (close(index) == OK)

printf("关闭操作成功\n");

printf("----------------------------------------------\n");

break;

case READ:

if (show\_openlist() == 0)

{

printf("当前没有文件被打开a\n");

printf("----------------------------------------------\n");

break;

}

printf("请输入读取文件的索引号：\n");

scanf("%d", &index);

printf("请输入想要读取文件长度：\n");

scanf("%d", &count);

if (read(index, 0, count) == OK)

printf("读文件操作成功\n");

printf("----------------------------------------------\n");

break;

case WRITE:

if (show\_openlist() == 0)

{

printf("当前没有文件被打开\n");

printf("----------------------------------------------\n");

break;

}

printf("请输入要写入文件的索引号：\n");

scanf("%d", &index);

printf("请输入想要写入文件长度：\n");

scanf("%d", &count);

if (write(index, 0, count) == OK)

printf("写入操作成功\n");

printf("----------------------------------------------\n");

break;

case LSEEK:

if (show\_openlist() == 0)

{

printf("当前没有文件被打开\n");

printf("----------------------------------------------\n");

break;

}

printf("请输入要写入文件的索引号:\n");

scanf("%d", &index);

printf("请输入想要设置的文件相对位置\n");

scanf("%d", &pos);

lseek(index, pos);

printf("----------------------------------------------\n");

break;

case OPLIST:

if (show\_openlist() == 0)

{

printf("\t\t\n当前没有文件被打开\n");

printf("----------------------------------------------\n");

break;

}

printf("----------------------------------------------\n");

break;

case HELP:

show\_help();

break;

case LDISK:

show\_ldisk();

break;

default:

printf("指令错误\n");

printf("----------------------------------------------\n");

break;

}

fflush(stdin);

Init\_block(ch, CH\_LENGTH);

}

}