

实验 4 文件系统设计

1. 实验目的

通过一个简单多用户文件系统的设计,加深理解文件系统的内部功能及内部实现。

2. 实验内容

为 Linux 系统设计一个简单的二级文件系统。要求做到以下几点:

(1) 可以实现下列几条命令(至少 4 条);

login	用户登录
dir	列文件目录
create	创建文件
delete	删除文件
open	打开文件
close	关闭文件
read	读文件
write	写文件

(2) 列目录时要列出文件名、物理地址、保护码和文件长度;

(3) 源文件可以进行读写保护。

3. 实验提示

(1) 首先应确定文件系统的数据结构:主目录、子目录及活动文件等。主目录和子目录都以文件的形式存放于磁盘,这样便于查找和修改。

(2) 用户创建的文件,可以编号存储于磁盘上。如 file0, file1, file2... 并以编号作为物理地址,在目录中进行登记。

实验 4 指导

【实验内容】

〈任务〉

为 Linux 系统设计一个简单的二级文件系统。要求做到以下几点：

1. 可以实现下列几条命令：

login	用户登录
dir	列目录
create	创建文件
delete	删除文件
open	打开文件
close	关闭文件
read	读文件
write	写文件

2. 列目录时要列出文件名,物理地址,保护码和文件长度

3. 源文件可以进行读写保护

〈程序设计〉

1. 设计思想

本文件系统采用两级目录,其中第一级对应于用户账号,第二级对应于用户账号下的文件。另外,为了简单本文件系统未考虑文件共享、文件系统安全以及管道文件与设备文件等特殊内容。对这些内容感兴趣的读者,可以在本系统的程序基础上进行扩充。

2. 主要数据结构

(1) i 节点

```
struct inode {
    struct inode *i_forw;
    struct inode *i_back;
    char i_flag;
    unsigned int i_ino;           /* 磁盘 i 节点标号 */
    unsigned int i_count;        /* 引用计数 */
    unsigned short di_number;    /* 关联文件数,当为 0 时,则删除该文件 */
    unsigned short di_mode;     /* 存取权限 */
    unsigned short di_uid;      /* 磁盘 i 节点用户 id */
    unsigned short di_gid;      /* 磁盘 i 节点组 id */
}
```

```
unsigned int di_addr [NADDR];      /* 物理块号 */
```

(2) 磁盘 i 节点

Struct dinode

```
{
    unsigned short di_number;      /* 关联文件数 */
    unsigned short di_mode;        /* 存取权限 */

    unsigned short di_uid;
    unsigned short di_gid;
    unsigned long di_size;         /* 文件大小 */
    unsigned int di_addr [NADDR]; /* 物理块号 */
}
```

(3) 目录项结构

Struct direct

```
{
    char d_name [DIRSIZ];          /* 目录名 */
    unsigned int d_ino;             /* 目录号 */
}
```

(4) 超级块

Struct filsys

```
{
    unsigned short s_isize;         /* i 节点块数 */
    unsigned long s_fsize;          /* 数据块块数 */

    unsigned int s_nfree;           /* 空闲块块数 */
    unsigned short s_pfree;         /* 空闲块指针 */
    unsigned int s_free [NICFREE]; /* 空闲块堆栈 */

    unsigned int s_ninode;          /* 空闲 i 节点数 */
    unsigned short s_pinode;        /* 空闲 i 节点指针 */
    unsigned int s_inode [NICINOD]; /* 空闲 i 节点数组 */
    unsigned int s_rinode;          /* 铭记 i 节点 */

    char s_fmod;                    /* 超级块修改标志 */
};
```

(5) 用户密码

Struct pwd

```
{
    unsigned short P_uid;
```

```

    unsigned short P_gid;
    char password [PWOSIZ];
};

```

(6) 目录

Struct dir

```

{
    struct direct direct [DIRNUM];
    int size;
};

```

(7) 查找内存 i 节点的 hash 表

Struct hinode

```

{
    struct inode *i_forw;
};

```

(8) 系统打开表

Struct file

```

{
    char f_flag;                /* 文件操作标志 */
    unsigned int f_count;        /* 引用计数 */
    struct inode *f_inode;       /* 指向内存 i 节点 */
    unsigned long f_off;         /* 读/写指针 */
};

```

(9) 用户打开表

Struct user

```

{
    unsigned short u_default_mode;
    unsigned short u_uid;        /* 用户标志 */
    unsigned short u_gid;        /* 用户组标志 */
    unsigned short u_ofile [NOFILE]; /* 用户打开表 */
};

```

3. 主要函数

- (1) i 节点内容获取函数 iget()(详细描述略)。
- (2) i 节点内容释放函数 iput()(详细描述略)。
- (3) 目录创建函数 mkdir()(详细描述略)。
- (4) 目录搜索函数 namei()(详细描述略)。
- (5) 磁盘块分配函数 balloc()(详细描述略)。
- (6) 磁盘块释放函数 bfree()(详细描述略)。
- (7) 分配 i 节点区函数 ialloc()(详细描述略)。

- (8) 释放 i 节点区函数 ifree()(详细描述略)。
- (9) 搜索当前目录下文件的函数 iname()(详细描述略)。
- (10) 访问控制函数 access()(详细描述略)。
- (11) 显示目录和文件用函数 _dir()(详细描述略)。
- (12) 改变当前目录用函数 chdir()(详细描述略)。
- (13) 打开文件函数 open()(详细描述略)。
- (14) 创建文件函数 create()(详细描述略)。
- (15) 读文件用函数 read()(详细描述略)。
- (16) 写文件用函数 write()(详细描述略)。
- (17) 用户登录函数 login()(详细描述略)。
- (18) 用户退出函数 logout()(详细描述略)。
- (19) 文件系统格式化函数 format()(详细描述略)。
- (20) 进入文件系统函数 install()(详细描述略)。
- (21) 关闭文件函数 close()(详细描述略)。
- (22) 退出文件系统函数 halt()(详细描述略)。
- (23) 文件删除函数 delete()(详细描述略)。

4. 主程序说明

Begin

- | | |
|--------|----------------------------|
| Step1 | 对磁盘进行格式化 |
| Step2 | 调用 install(),进入文件系统 |
| Step3 | 调用 _dir(),显示当前目录 |
| Step4 | 调用 login(),用户注册 |
| Step5 | 调用 mkdir()和 chdir()创建目录 |
| Step6 | 调用 creat(),创建文件 0 |
| Step7 | 分配缓冲区 |
| Step8 | 写文件 0 |
| Step9 | 关闭文件 0 和释放缓冲 |
| Step10 | 调用 mkdir()和 chdir()创建子目录 |
| Step11 | 调用 creat(),创建文件 1 |
| Step12 | 分配缓冲区 |
| Step13 | 写文件 1 |
| Step14 | 关闭文件 1 和释放缓冲 |
| Step15 | 调用 chdir 将当前目录移到上一级 |
| Step16 | 调用 creat(),创建文件 2 |
| Step17 | 分配缓冲区 |
| Step18 | 调用 write(),写文件 2 |
| Step19 | 关闭文件 2 和释放缓冲 |
| Step20 | 调用 delete(),删除文件 0 |
| Step21 | 调用 creat(),创建文件 3 |
| Step22 | 为文件 3 分配缓冲区 |
| Step23 | 调用 write(),写文件 3 |

Step24	关闭文件 3 并释放缓冲区
Step25	调用 open(), 打开文件 2
Step26	为文件 2 分配缓冲
Step27	写文件 3 后关闭文件 3
Step28	释放缓冲
Step29	用户退出 (logout)
Step30	关闭 (halt)

End

由上述描述过程可知,该文件系统实际是为用户提供一个解释执行相关命令的环境。主程序中的大部分语句都被用来执行相应的命令。

下面,我们给出每个过程的相关 C 语言程序。读者也可以使用这些子过程,编写出一个用 Shell 控制的文件系统界面。

〈程序〉

1. 编程管理文件 makefile

本文件系统程序用 makefile 编程管理工具进行管理。其内容如下。

```

***** /

/ *****
makefile
***** /

filsys: main.o igetput.o iallfre.o ballfre.o name.o access.o log.o close.o creat.o delete.o dir.o
      open.o rdwt.o format.o install.o halt.o cc -o filsys main.o igetput.o iallfre.o ballfre.o
      name.o access.o log.o close.o creat.o delete.o dir.o open.o rdwt.o format.o install.o halt.o
main.o: main.c filesys.h
      cc -c main.c
igetput.o: igetput.c filesys.h
      cc -c igetput.c
iallfre.o: iallfre.c filesys.h
      cc -c iallfre.c
ballfre.o: ballfre.c filesys.h
      cc -c ballfre.c
name.o: name.c filesys.h
      cc -c name.c
access.o: access.c filesys.h
      cc -c access.c
log.o: log.c filesys.h
      cc -c log.c
close.o: close.c filesys.h
      cc -c close.c
creat.o: creat.c filesys.h
      cc -c creat.c

```

```

delete.o: delete.c filesys.h
    cc -c delete.c
dir.o: dir.c filesys.h
    cc -c dir.c
open.o: open.c filesys.h
    cc -c open.c
rdwt.o: rdwt.c filesys.h
    cc -c rdwt.c
format.o: format.c filesys.h
    cc -c format.c
install.o: install.c filesys.h
    cc -c install.c
halt.o: halt.c
    cc -c halt.c

```

2. 头文件 filesys.h

头文件 filesys.h 用来定义本文件系统中所使用的各种数据结构和常数符号。

```

/ *****
filesys.h

    定义本文件系统的数据结构和常数
***** /

#define BLOCKSIZ 512
#define SYSOPENFILE 40
#define DIRNUM 128
#define DIRSIZ 14
#define PWDSIZ 12
#define PWDNUM 32
#define NOFILE 20
#define NADDR 10
#define NHINO 128 /* must be power of 2 */
#define USERNUM 10
#define DINODESIZ 32

/* filsys */
#define DINODEBLK 32
#define FILEBLK 512
#define NICFREE 50
#define NICINOD 50
#define DINODESTART 2 * BLOCKSIZ
#define DATASTART (2 + DINODEBLK) * BLOCKSIZ

```

```

/* di_mode */
#define DIEMPTY          00000

#define DIFILE           01000
#define DIDIR            02000

#define UDIREAD          00001 /* user */
#define UDIWRITE         00002
#define UDIEXECUTE       00004
#define GDIREAD          00010 /* group */
#define GDIWRITE         00020
#define GDIEXECUTE       00040
#define ODIREAD          00100 /* other */
#define ODIWRITE         00200
#define ODIEXECUTE       00400

#define READ             1
#define WRITE            2
#define EXECUTE          3

#define DEFAULTMODE      00777

/* i_flag */
#define IUPDATE 00002

/* s_fmod */
#define SUPDATE 00001

/* f_flag */
#define FREAD          00001
#define FWRITE         00002
#define FAPPEND        00004

/* error */
#define DISKFULL 65535

/* fseek origin */
#define SEEK_SET 0

/* 文件系统 数据结构 */
struct inode {
    struct inode *i_forw;
    struct inode *i_back;

```



```

char i_flag;
unsigned int i_ino;    /* 磁盘i节点标志 */
unsigned int i_count;  /* 引用计数 */
unsigned short di_number; /* 关联文件数。当为0时,则删除该文件 */
unsigned short di_mode; /* 存取权限 */
unsigned short di_uid;
unsigned short di_gid;
unsigned short di_size; /* 文件大小 */
unsigned int di_addr [NADDR]; /* 物理块号 */
};

struct dinode {
    unsigned short di_number; /* 关联文件数 */
    unsigned short di_mode; /* 存取权限 */
    unsigned short di_uid;
    unsigned short di_gid;
    unsigned long di_size; /* 文件大小 */
    unsigned int di_addr [NADDR]; /* 物理块号 */
};

struct direct {
    char d_name [DIRSIZ];
    unsigned int d_ino;
};

struct filsys {
    unsigned short s_isize; /* i节点块数 */
    unsigned long s_fsize; /* 数据块数 */
    unsigned int s_nfree; /* 空闲块 */
    unsigned short s_pfree; /* 空闲块指针 */
    unsigned int s_free [NICFREE]; /* 空闲块堆栈 */

    unsigned int s_ninode; /* number of free inode in s_inode */
    unsigned short s_pinode; /* pointer of the sinode */
    unsigned int s_inode [NICINOD]; /* 空闲i节点数组 */
    unsigned int s_rinode; /* remember inode */

    char s_fmod; /* 超级块修改标志 */
};

struct pwd {
    unsigned short p_uid;
    unsigned short p_gid;

```

```

        char password [PWDSIZ];
    };

    struct dir {
        struct direct direct [DIRNUM];
        int size;    /* 当前目录大小 */
    };

    struct hinode {
        struct inode * i_forw;    /* hash 表指针 */
    };

    struct file {
        char f_flag;    /* 文件操作标志 */
        unsigned int f_count;    /* 引用计数 */

        struct inode * f_inode;    /* 指向内存 i 节点 */
        unsigned long f_off;    /* read/write character pointer */
    };

    struct user {
        unsigned short u_default-mode;
        unsigned short u-uid;
        unsigned short u-gid;
        unsigned short u-ofile [NOFILE];    /* 用户打开文件表 */
        /* system open file pointer number */
    };

    /* 下为全局变量 */
    extern struct hinode hinode [NHINO];
    extern struct dir dir;    /* 当前目录(在内存中全部读入) */
    extern struct file sys_ofile [SYSOPENFILE];
    extern struct filsys filsys;    /* 内存中的超级块 */
    extern struct pwd pwd [PWDNUM];
    extern struct user user [USERNUM];
    extern FILE * fd;    /* the file system column of all the system */
    extern struct inode * cur-path-inode;
    extern int user-id;

    /* prototype of the sub routine used in the file system */
    extern struct inode * iget( );
    extern iput( );
    extern unsigned int balloc( );
    extern bfree( );

```

```

extern struct inode * ialloc( );
extern ifree( );
extern unsigned int namei( );
extern unsigned short iname( );
extern unsigned int access( );
extern _dir( )
extern mkdir( );
extern chdir( );
extern unsigned short open( );
extern creat( );
extern unsigned int read( );
extern unsigned int write( );
extern int login( );
extern logout( );
extern install( );
extern format( );
extern close( );
extern halt( );

```

3. 主程序 main()(文件名 main.c)

主程序 main.c 用来测试文件系统的各种设计功能,其主要功能描述如程序设计中的第 4 部分。

程序:

```

#include <stdio.h>
#include "fileSYS.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;

main( )
{
    unsigned short ab_fd1, ab_fd2, ab_fd3, ab_fd4;
    unsigned short bhy_fd1;

```

```

char * buf;

printf ("\nDo you want to format the disk \n");
if (getchar( )=='y')
    printf("\nFormat will erase all context on the disk \nAre You Sure!! \n");
if (getchar( )=='y')
    format( );

install( );
_dir( );

login(2118, "abcd");
user_id=0;

mkdir("a2118");
chdir("a2118");
wj_fd1=creat(2118,"ab_file0.c", 01777);
buf = (char * ) malloc (BLOCKSIZ * 6+5);
write(ab_fd1, buf, BLOCKSIZ * 6+5);
close(user_id, ab_fd1);
free(buf);

mkdir("subdir");
chdir("subdir");
wj_fd2=creat(2118,"file1.c", 01777);
buf=(char * ) malloc (BLOCKSIZ * 4+20);
write (ab_fd2, buf, BLOCKSIZ * 4+20);
close(user_id, ab_fd2);
free(buf);
chdir("../");
ab_fd3=creat(2118,"_file2.c", 01777);
buf=(char * ) malloc (BLOCKSIZ * 10+255);
write(ab_fd3, buf, BLOCKSIZ * 3+255);
close(ab_fd3);
free(buf);

delete("ab_file0.c");

ab_fd4=creat(2118, "ab_file3.c", 01777);
buf=(char * ) malloc (BLOCKSIZ * 8+300);
write (ab_fd4, buf, BLOCKSIZ * 8+300);
close(ab_fd4);
free(buf);

```

```

ab_fd3=open(2118, "ab_file2.c", FAPPEND);
buf=(char *) malloc (BLOCKSIZ * 3+100);
write (ab_fd3, buf, BLOCKSIZ * 3+100);
close (ab_fd3);

free (buf);

_dir( );
chdir (".");
logout( );
halt( );
}

```

4. 初始化磁盘格式程序 format()(文件名 format.c)

```

#include <stdio.h>
#include "fileys.h"

format( )
{
    struct inode *inode;
    struct direct dir_buf [BLOCKSIZ / (DIRSIZ+2)];
    struct pwd passwd [BLOCKSIZ/(PWDSIZ+4)];
    /*
                                {
                                {2116, 03, "dddd"},
                                {2117, 03, "bbbb"},
                                {2118, 04, "abcd"},
                                {2119, 04, "cccc"},
                                {2220, 05, "eeee"},
                                };

    */
    struct filsys;
    unsigned int block_buf [BLOCKSIZ / sizeof (int)];
    char *buf;
    int i, j;
    /* creat the file system file */
    fd = fopen ("filesystem", "r+w+b");
    buf = (char *) malloc ((DINODEBLK+FILEBLK+2) * BLOCKSIZ * sizeof(char));
    if (buf==NULL)
    {

```

• 114 •

```

    printf ("\nfile system file creat failed!!! \n");
    exit (0);
}
fseek (fd, 0, SEEK_SET);

fwrite(buf,1, (DINODEBLK+FILEBLK+2) * BLOCKSIZ * sizeof(char),fd);

/* 0. initialize the passwd */
passwd[0].p_uid=2116; passwd[0].p_gid=03;
strcpy(passwd[0].password, "dddd");
passwd[1].p_uid=2117; passwd[1].p_gid=03;
strcpy(passwd[1].password, "bbbb");
passwd[2].p_uid=2118; passwd[2].p_gid=04;
strcpy(passwd[2].password, "abcd");
passwd[3].p_uid=2119; passwd[3].p_gid=04;
strcpy(passwd[3].password, "cccc");
passwd[4].p_uid=2220; passwd[4].p_gid=05;
strcpy(passwd[4].password, "eeee");

/* 1. creat the main directory and its sub dir etc and the file password */

inode =iget(0);    /* 0 empty dinode id */
inode->di_mode = DIEMPTY;
iput(inode);

inode=iget(1);    /* 1 main dir id */
inode->di_number=1;
inode->di_mode=DEFAULTMODE | DIDIR;
inode->di_size=3 * (DIRSIZ+2);
inode->di_addr[0]=0;    /* block 0# is used by the main directory */
strcpy(dir_buf[0].d_name, ".");
dir_buf[0].d_ino=1;
strcpy(dir_buf[1].d_name, ".");
dir_buf[1].d_ino=1;
strcpy(dir_buf[2].d_name, "etc");
dir_buf[2].d_ino=2;
fseek(fd, DATASTART, SEEK_SET);
fwrite(dir_buf, 1, 3 * (DIRSIZ+2), fd);
iput(inode);

inode=iget(2);    /* 2 etc dir id */
inode->di_number=1;

```

```

inode->di_mode=DEFAULTMODE | DIDIR;
inode->di_size=3*(DIRSIZ+2);
inode->di_addr[0]=1; /* block 1# is used by the etc directory */
strcpy(dir_buf[0].d_name, "..");
dir_buf[0].d_ino=1;
strcpy(dir_buf[1].d_name, ".");
dir_buf[1].d_ino=2;
strcpy(dir_buf[2].d_name, "password");
dir_buf[2].d_ino=3;
fseek(fd, DATASTART+BLOCKSIZ*1, SEEK_SET);
fwrite(dir_buf, 1*3*(DIRSIZ+2), fd);
input(inode);

inode=iget(3); /* 3 password id */
inode->di_number=1;
inode->di_mode=DEFAULTMODE | DIFILE;
inode->di_size=BLOCKSIZ;
inode->di_addr[0]=2; /* block 2# is used by the password file */
for (i=5; i<PWDNUM; i++)
{
    passwd[i].p_uid=0;
    passwd[i].p_gid=0;
    strcpy(passwd[i].password, " ");
}
fseek(fd, DATASTART+2*BLOCKSIZ, SEEK_SET);

fwrite(passwd, 1, BLOCKSIZ, fd);
input(inode);

/* 2. initialize the superblock */

filsys.s_isize=DINODEBLK;
filsys.s_fsize=FILEBLK;

filsys.s_ninode=DINODEBLK*BLOCKSIZ/DINODESIZ-4;
filsys.s_nfree=FILEBLK-3;

for (i=0; i<NICINOD; i++)
{
    /* begin with 4, 0,1,2,3, is used by main, etc, password */
    filsys.s_inode[i]=4+i;
}

```

```

    filsys.s_pinode=0;
    filsys.s_rinode=NICINOD+4;

    block_buf[NICFREE-1]=FILEBLK+1;    /* FILEBLK+1 is a flag of end */
    for (i=0; i<NICFREE-1; i++)
        block_buf[NICFREE-2-i]=FILEBLK-i;
    fseek (fd, DATASTART+BLOCKSIZ*(FILEBLK-NICFREE-1), SEEK_SET);
    fwrite (block_buf, 1, BLOCKSIZ, fd);

    for (i=FILEBLK-NICFREE-1; i>2; i-=NICFREE)
    {
        for (j=0; j<NICFREE; j++)
        {
            block_buf[j]=i-j;
        }
        fseek(fd,DATASTART+BLOCKSIZ*(i-1), SEEK_SET);
        fwrite(block_buf, 1, BLOCKSIZ, fd);
    }

    j=1;
    for (i=i; i>2; i--)
    {
        filsys.s_free[NICFREE+i-j]=i;
    }

    filsys.s_pfree=NICFREE-j;
    filsys.s_pinode=0;

    fseek(fd, BLOCKSIZ, SEEK_SET);
    fwrite(&filsys, 1, sizeof(struct filsys), fd);
}

```

5. 进入文件系统程序 install()(文件名 install.c)

```

#include <stdio.h>
#include <string.h>
#include "filesys.h"

install( )
{
    int i,j;

    /* 0. open the file column */

```



```

fd=fopen("filesystem", "w+r+b");
if (fd=NULL)
{
    printf("\nfileys can not be loaded\n");
    exit(0);
}

/* 1. read the filsys from the superblock */
fseek (fd, BLOCKSIZ, SEEK-SET);
fwrite(&filsys, 1, sizeof(struct filsys), fd);

/* 2. initialize the inode hash chain */
for (i=0; i<NHINO; i++)
{
    hinode[i].i_forw=NULL;
}

/* 3. initialize the sys_ofile */
for (i=0; i<SYSOPENFILE; i++)
{
    sys_ofile[i].f_count=0;
    sys_ofile[i].f_inode=NULL;
}

/* 4. initialize the user */
for (i=0; i<USERNUM; i++)
{
    user[i].u_uid=0;
    user[i].u_gid=0;
    for (j=0; j<NOFILE; j++)
    {
        user[i].u_ofile[j]=SYSOPENFILE+1;
    }
}

/* 5. read the main directory to initialize the dir */
cur_path_inode=iget(1);
dir.size=cur_path_inode->di_size/(DIRSIZ+2);
for(i=0; i<DIRNUM; i++)
{
    strcpy(dir.direct[i].d_name, "          ");
    dir.direct[i].d_ino=0;
}

```

```

for (i=0; i<dir.size/(BLOCKSIZ/(DIRSIZ+2)); i++)
{
    fseek(fd, DATASTART+BLOCKSIZ*cur_path_inode->di_addr[i], SEEK_SET);
    fread(&dir.direct[(BLOCKSIZ/(DIRSIZ+2))*i], 1, BLOCKSIZ, fd);
}
fseek(fd, DATASTART+BLOCKSIZ*cur_path_inode->di_addr[i], SEEK_SET);
fread(&dir.direct[(BLOCKSIZ/(DIRSIZ+2))*i], 1,
      cur_path_inode->di_size % BLOCKSIZ, fd);
}

```

6. 退出程序 halt() (文件名 halt.c)

```

#include <stdio.h>
#include "filesys.h"

halt()
{
    struct inode *inode;
    int i, j;

    /* 1. write back the current dir */
    chdir ("..");
    iput(cur_path_inode);

    /* 2. free the u_ofile and sys_ofile and inode */
    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;
                }
            }
        }
    }

    /* 3. write back the filesys to the disk */
    fseek (fd, BLOCKSIZ, SEEK_SET);
    fwrite (&filsys, 1, sizeof(struct filsys), fd);
}

```

```

/* 4. close the file system column */
fclose (fd);

/* 5. say GOOD BYE to all the user */
printf ("\nGood Bye. See You Next Time. Please turn off the switch\n");
exit (0);

```

7. 获取释放 i 节点内容程序 iget()/iput()(文件名 igetputc)

```

#include <stdio.h>
#include "fileys.h"

struct inode * iget (dinodeid)    /* iget( ) */
unsigned int dinodeid;
{
    int existed=0, inodeid;
    long addr;
    struct inode * temp, * newinode;

    inodeid=dinodeid % NHINO;
    if (hinode [inodeid]. i_forw == NULL) existed = 0;
    else
    {
        temp=hinode [inodeid]. i_forw;
        while (temp)
        {
            if (temp->i_ino == inodeid)
                /* existed */
                {
                    existed=1;
                    temp->i_count ++;
                    return temp;
                }
            /* not existed */
            else
                temp =temp->i_forw;
        }
    }

    /* not existed */
    /* 1. calculate the addr of the dinode in the file sys column */
    addr = DINODESTART + dinodeid * DINODESIZ;
    /* 2. malloc the new inode */
    newinode =(struct inode * ) malloc (sizeof (struct inode));

```

```

    /* 3. read the dinode to the inode */
    fseek (fd, addr, SEEK_SET);
    fread(&(newinode->di_number), DINODESIZ, 1, fd);
    /* 4. put it into hinode [inodeid] queue */
    newinode->i_forw=hinode[inodeid].i_forw;
    newinode->i_back=newinode;
    newinode->i_forw->i_back=newinode;
    hinode[inodeid].i_forw=newinode;
    /* 5. initialize the inode */
    newinode->i_count=1;
    newinode->i_flag=0;    /* flag for not update */
    newinode->i_ino=dinodeid;

    return newinode;
}

iput(pinode)          /* iput ( ) */
struct inode *pinode;
{
    long addr;
    unsigned int block_num;
    int i;

    if (pinode->i_count>1)
    {
        pinode->i_count--;
        return;
    }
    else
    {
        if (pinode->di_number != 0)
        {
            /* write back the inode */
            addr =DINODESTART + pinode->i_ino * DINODESIZ;
            fseek(fd, addr, SEEK_SET);
            fwrite(&pinode->di_number, DINODESIZ,1,fd);
        }
        else
        {
            /* rm the inode & the block of the file in the disk */
            block_num=pinode->di_size/BLOCKSIZ;
            for (i=0; i<block_num; i++)

```

```

        {
            balloc(pinode->di_addr[i]);
        }
        ifree(pinode->i_ino);
    };

    /* free the inode in the memory */
    if (pinode->i_forw == NULL)
        pinode->i_back->i_forw = NULL;
    else
    {
        pinode->i_forw->i_back = pinode->i_back;
        pinode->i_back->i_forw = pinode->i_forw;
    };
    free (pinode);
};
}

```

8. i 节点分配和释放函数 ialloc()和 ifree()(文件名 iallfre.c)

```

#include <stdio.h>
#include "fileys.h"

static struct dinode block_buf [BLOCKSIZ/DINODESIZ];

struct inode * ialloc ( )          /* ialloc */
{
    struct inode * temp_inode;
    unsigned int cur_di;
    int i, count, block_end_flag;

    if (fileys.s_pinode == NICINOD)    /* s_inode empty */
    {
        i=0;
        count = 0;
        block_end_flag=1;
        fileys.s_pinode=NICINOD-1;
        cur_di=fileys.s_rinode;
        while ((count <NICINOD) && (count <= fileys.s_ninode))
        {
            if (block_end_flag)

                {

```

```

        fseek (fd,DINODESTART+cur_di * DINODESIZ);
        fread (block_buf, 1, BLOCKSIZ, fd);
        block_end_flag=0;
        i=0;
    }
    while (block_buf[i].di_mode == DIEMPTY)
    {
        cur_di ++;
        i ++;
    }
    if (i==NICINOD)
        block_end_flag=1;
    else
    {
        filsys.s_inode[filsys.s_pinode--]=cur_di;
        count ++;
    }
}
    filsys.s_rinode=cur_di;
}
temp_inode=iget(filsys.s_inode [filsys.s_pinode]);
fseek (fd, DINODESTART+filsys.s_inode [filsys.s_pinode] * DINODESIZ, SEEK_SET);
fwrite (&temp_inode->di_number, 1, sizeof (struct dinode), fd);
filsys.s_pinode ++;
filsys.s_ninode --;
filsys.s_fmod=SUPDATE;
return temp_inode;
}

```

```

ifree(dinodeid)          /* ifree */
unsigned dinodeid;
{
    filsys.s_ninode ++;
    if (filsys.s_pinode != NICINOD)    /* not full */
    {
        filsys.s_inode[filsys.s_pinode]=dinodeid;
        filsys.s_pinode ++;
    }
    else /* full */
    {
        if (dinodeid < filsys.s_rinode)
        {
            filsys.s_inode [NICINOD]=dinodeid;

```

```

        filsys.s_rinode=dinodeid;
    }
}
}

```

9. 磁盘块分配与释放函数 balloc()与 bfree()(文件名 ballfre.c)

```

#include <stdio.h>
#include "filesys.h"

static unsigned int block_buf[BLOCKSIZ];

unsigned int balloc( )
{
    unsigned int free_block, free_block_num;
    int i;

    if (filsys.s_nfree==0)
    {
        printf ("\nDisk Full!!! \n");
        return DISKFULL;
    };

    free_block=filsys.s_free[filsys.s_pfree];
    if (filsys.s_pfree==NICFREE-1)
    {
        fread(block_buf, 1, BLOCKSIZ, fd);
        free_block_num=block_buf[NICFREE]; /* the total block num in the group */
        for (i=0; i<free_block_num; i++)
        {
            filsys.s_free[NICFREE-1-i]=block_buf[i];
        }
        filsys.s_pfree=NICFREE-free_block_num;
    }
    else filsys.s_pfree++;

    filsys.s_nfree--;
    filsys.s_fmod=SUPDATE;

    return free_block;
}

bfree (block_num)

```

```

unsigned int block_num;
{
    int i;

    if (filsys.s_pfree==0)    /* s_free full */
    {
        block_buf[NICFREE]=NICFREE;
        for (i=0; i<NICFREE; i++)
        {
            block_buf[i]=filsys.s_free[NICFREE-1-i];
        }
        filsys.s_pfree=NICFREE-1;
    }

    fwrite(block_buf, 1, BLOCKSIZ, fd);
    filsys.s_nfree++;
    filsys.s_fmod=SUPDATE;
}

```

10. 搜索函数 namei()和 iname()(文件名 name.c)

```

#include <string.h>
#include <stdio.h>
#include "filesys.h"

```

```

unsigned int namei (name)                /* namei */
char * name;
{
    int i, notfound=1;

    for (i=0; ((i<dir.size)&&(notfound));i++)
        if ((! strcmp(dir.direct[i].d_name, name)) && (dir.direct[i].d_ino != 0))
            return i;    /* find */
    /* not find */
    return NULL;
}

```

```

unsigned short iname (name)              /* iname */
char * name;
{
    int i, notfound=1;

    for (i=0; ((i<DIRNUM)&&(notfound)); i++)

```



```

        if (dir.direct[i].d_ino==0)
        {
            notfound=0;
            break;
        }
    if (notfound)
    {
        printf("\nThe current directory is full!!! \n");
        return 0;
    }
    else
    {
        strcpy(name, dir.direct[i].d_name);
        return i;
    }
}

```

11. 访问控制函数 access()(文件名 access.c)

```

#include <stdio.h>
#include "fileys.h"

unsigned int access (user_id, inode, mode)
unsigned int user_id;
struct inode *inode;
unsigned short mode;
{

    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 0;
}
}

```

12. 显示列表函数 `dir()` 和目录创建函数 `mkdir()` 等(文件名 `dir.c`)

```

#include <stdio.h>
#include <string.h>
#include "fileys.h"

_dir()          /* _dir */
{
    unsigned int di_mode;
    int i, one;
    struct inode *temp_inode;

    printf("\n CURRENT DIRECTORY :\n");
    for (i=0; i<dir.size; i++)
    {
        if (dir.direct[i].d_ino != DIEMPTY)
        {
            printf("%DIRSIZs", dir.direct[i].d_name);
            temp_inode=iget(dir.direct[i].d_ino);
            di_mode=temp_inode->di_mode;
            for (i=0; i<9; i++)
            {
                one =di_mode % 2;
                di_mode=di_mode /2;

                if (one) printf ("x");
                else printf ("-");
            }
            if (temp_inode->di_mode && DIFILE==1)

```



```

buf[1].d_ino = cur_path.inode->i_ino;

block=ballocc( );
fseek(fd, DATASTART+block * BLOCKSIZ, SEEK_SET);
fwrite(buf,1,BLOCKSIZ,fd);

inode->di_size=2 * (DIRSIZ+2);
inode->di_number=1;
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_addr[0]=block;

iput(inode);

return;
}

chdir (dirname)                /* chdir */
char * dirname;
{
    unsigned int dirid;
    struct inode * inode;
    unsigned short block;
    int i,j,low=0, high=0;

    dirid=namei(dirname);
    if (dirid==NULL)
    {
        printf("\n%s does not existed\n", dirname);
        return;
    }
    inode=iget (dirid);
    if (! access (user_id, inode, user[user_id].u_default_mode))
    {
        printf("\nhas not access to the directory %s", dirname);
        iput(inode);
        return;
    }

    /* pack the current directory */
    for (i=0; i<dir.size; i++)
    {

```

```

        for (j<DIRNUM;j++)
            if (dir.direct[j].d_ino==0) break;
        memcpy(&dir.direct[i], &dir.direct[j], DIRSIZ+2);
        dir.direct[j].d_ino=0;
    }
    /* write back the current directory */
    for (i=0; i<cur_path_inode->di_size/BLOCKSIZE+1; i++)
    {
        bfree (cur_path_inode->di_addr[i]);
    }
    for (i=0; i<dir.size; i+=BLOCKSIZE/(DIRSIZ+2))
    {
        block=ballocc( );
        cur_path_inode->di_addr[i]=block;
        fseek(fd, DATASTART+block*BLOCKSIZE, SEEK_SET);
        fwrite(&dir.direct[i], 1, BLOCKSIZE, fd);
    }
    cur_path_inode->di_size=dir.size*(DIRSIZ+2);
    iput(cur_path_inode);

    cur_path_inode=inode;
    /* read the change dir from disk */
    j=0;
    for (i=0; i<inode->di_size/BLOCKSIZE+1; i++)
    {
        fseek(fd,DATASTART+inode->di_addr[i]*BLOCKSIZE, SEEK_SET);
        fread(&dir.direct[j], 1, BLOCKSIZE, fd);
        j+=BLOCKSIZE/(DIRSIZ+2);
    }

    return;

}

```

13. 文件创建函数 creat()(文件名 creat.c)

```

#include <stdio.h>
#include "fileys.h"

creat (user_id, filename, mode)
unsigned int user_id;
char * filename;
unsigned short mode;

```

```

{
    unsigned int di_ith, di_ino;
    struct inode *inode;
    int i,j;

    di_ino=namei(filename);
    if (di_ino != NULL) /* already existed */
    {
        inode=iget(di_ino);
        if (access (user_id, inode, mode)==0)
        {
            iput (inode);
            printf ("\creat access not allowed \n");
            return;
        }
        /* free all the block of the old file */
        for (i=0; i<inode->di_size / BLOCKSIZ+1; i++)
        {
            bfree (inode->di_addr[i]);
        }
        /* to do: add code here to update the pointer of the sys_file */
        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; j++)
                    if (sys_ofile [j].f_count=0)
                    {
                        user [user_id].u_ofile[i]=j;
                        sys_ofile[j].f_flag=mode;
                    }
                return i;
            }
    }
    else /* not existed before */
    {
        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=mode;
    sys_ofile[i].f_count=0;
    sys_ofile[i].f_off=0;
    sys_ofile[i].f_inode=inode;

    return j;
}

}

```

14. 打开文件函数 open() (文件名 open.c)

```

#include <stdio.h>
#include "fileys.h"

unsigned short open(user_id, filename, openmode)
int user_id;
char * filename;
unsigned short openmode;

```

```

{
    unsigned int dinodeid;
    struct inode *inode;
    int i,j;

    dinodeid=namei(filename);
    if (dinodeid != NULL) /* no such file */
    {
        printf ("\nfile does not existed!!! \n");
        return NULL;
    }
    inode=iget(dinodeid);
    if (! access(user_id, inode, openmode)) /* access denied */
    {
        printf ("\nfile open has not access!!!");
        iput(inode);
        return NULL;
    }

    /* alloc the sys..ofile item */
    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 NULL;
    }
    sys_ofile[i].f_inode=inode;
    sys_ofile[i].f_flag=openmode;
    sys_ofile[i].f_count=1;
    if (openmode & FAPPEND)
        sys_ofile[i].f_off=inode->di_size;
    else
        sys_ofile[i].f_off=0;

    /* alloc the user open file item */
    for (j=0; j<NOFILE; j++)
        if (user[user_id].u_ofile[j]==0) break;
    if (j==NOFILE)
    {
        printf ("\nuser open file too much!!! \n");
        sys_ofile[i].f_count=0;
    }
}

```



```

        iput(inode);
        return NULL;
    }
    user[user_id].u_ofile[i]=1;

    /* if APPEND, free the block of the file before */
    if (openmode & FAPPEND)
    {
        for (i=0; i<inode->di_size /BLOCKSIZ +1; i++)
            bfree (inode->di_addr[i]);
        inode->di_size=0;
    }
    return j;
}

```

15. 关闭文件系统函数 close()(文件名 close.c)

```

#include <stdio.h>
#include "fileys.h"

close (user_id, cfd)          /* close */
unsigned int user_id;
unsigned short cfd;
{
    struct inode *inode;

    inode=sys_ofile [user[user_id].u_ofile[cfd]].f_inode;
    iput (inode);
    sys_ofile [user[user_id].u_ofile[cfd]].f_count --;
    user [user_id].u_ofile [cfd]=SYSOPENFILE + 1;
}

```

16. 删除文件函数 delete()(文件名 delete.c)

```

#include <stdio.h>
#include "fileys.h"

delete (filename)
char * filename;
{
    unsigned int dinodeid;
    struct inode *inode;

```

```

    dinodeid=namei(filename);
    if (dinodeid != NULL)
        inode = iget(dinodeid);
    inode->di_number--;
    iput(inode);
}

```

17. 读写文件函数 read()与 write()(文件名 rdwt.c)

```

#include <stdio.h>
#include "fileys.h"

unsigned int read (fd, buf, size)
int fd;
char * buf;
unsigned int size;
{
    unsigned long off;
    int block, block_off, i, j;
    struct inode * inode;
    char * temp_buf;

    inode=sys_ofile[user[user_id].u_ofile[fd]].f_inode;
    if (! (sys_ofile[user[user_id].u_ofile[fd]].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[fd]].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<(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;
    }

    block_off = (size-block_off) % BLOCKSIZ;
    block=inode->di_addr[off+size/BLOCKSIZ+1];
    fseek(fd, DATASTART+block*BLOCKSIZ, SEEK_SET);
    fread(temp_buf, 1, block_off, fd);

    sys_ofile[user[user_id].u_ofile[fd]].f_off += size;

    return size;
}

unsigned int write (fd, buf, size)                /* write */
int fd;
char * buf;
unsigned int size;
{
    unsigned long off;
    int block, block_off, i, j;
    struct inode * inode;
    char * temp_buf;

    inode=sys_ofile[user[user_id].u_ofile[fd]].f_inode;

    if (! (sys_ofile[user[user_id].u_ofile[fd]].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[fd]].f_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);
    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<=(size-block_off)/BLOCKSIZ; 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;
}

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[fd]].f_off += size;

return size;
}

```

18. 注册和退出函数 login() 和 logout() (文件名 log.c)

```

#include <stdio.h>
#include "fileSYS.h"

```

```

int login (uid, passwd)
unsigned short uid;

```

```

char *passwd;
{
    int i,j;

    for (i=0; i<PWDNUM; i++)
    {
        if ((uid==pwd[i].p_uid)&&(strcmp(passwd,pwd[i].password)))
        {
            for (j=0; j<USERNUM; i++)
                if (user[j].u_uid==0) break;
            if (j==USERNUM)
            {
                printf("\ntoo much user in the system, waited to login\n");
                return 0;
            }
            else
            {
                user[j].u_uid=uid;
                user[j].u_gid=pwd[i].p_gid;
                user[j].u_default_mode=DEFAULTMODE;

            }
            break;
        }
    }
    if (i==PWDNUM)
    {
        printf("\nincorrect password\n");
        return 0;
    }
    else
        return 1;
}

int logout (uid)                                /* logout */
unsigned short uid;
{
    int i,j,sys_no;
    struct inode *inode;

    for (i=0; i<USERNUM; i++)
        if (uid == user[i].u_uid) break;

```

```

if (i==USERNUM)
{
    printf("\nno such a file\n");
    return NULL;
}

for (j=0; j<NOFILE; j++)
{
    if (user[i].u_ofile[j] != SYSOPENFILE+1)
    {
        /* input the inode free the sys_ofile and clear the user_ofile */
        sys_no=user[i].u_ofile[j];
        inode=sys_ofile[sys_no].f_inode;
        input(inode);
        sys_ofile[sys_no].f_count --;
        user[i].u_ofile[j]=SYSOPENFILE+1;
    }
}

return 1;
}

```

[结果]

对上述 makefile 文件进行编译后可得执行文件“filsys”。在 Linux 或 UNIX System V 以上版本环境下,运行 filsys,可对上述文件系统程序进行测试。其结果如下:

```

/ *****
result;
the output of the filesystem run by the test program
***** /

$ filsys <CR>

$ format
$ Do you want to format the disk?    y
$ Format will erase all context on the disk. Are You Sure!!    y

$ install

$ dir
CURRENT DIRECTORY;
.          d xxx xxx xxx <dir> block chain;1
..         d xxx xxx xxx <dir> block chain;2

```

```

$ login
  please input your uid,2118
  password:
  ok! 2118's user id is, 0

$ mkdir a2118
$ chdir a2118

$ creat(2118, "file0.c", 01700)
  the file file0.c fd:0
$ write (0, buf, 3077)
$ close (0,0)

$ mkdir subdir
$ chdir subdir

$ creat(2118, "file1.c", 01700)
  the file file1.c fd:0
$ write (0, buf, 2068)

$ chdir ..

$ creat(2118, "file2.c", 01700)
  the file file2.c fd:1
$ write(1,buf,1791)

$ dir
  CURRENT DIRECTORY,
    .                d xxx xxx xxx <dir>block chain:1
    ..               d xxx xxx xxx <dir>block chain:2
    file0.c          f xxx --- --- 3077    block chain:4 5 6 7 8 9 10
    subdir           d xxx xxx xxx <dir>blokc chain:11
    file2.c          f xxx --- --- 1791    block chain: 17 18 19 20

$ delete file0.c

$ creat(2118, "file3.c", 01700)
  the file file3.c fd:2
$ write (2, buf, 4396)
$ close (0,2)

```

```
$ dir
CURRENT DIRECTORY:
.          d xxx xxx xxx <dir>block chain,1
..         d xxx xxx xxx <dir>block chain,2
subdir     d xxx xxx xxx <dir>block chain,11
file2.c    f xxx --- --- 1791    block chain, 17 18 19 20
file3.c    f xxx --- --- 4396    block chain,4 5 6 7 8 9 10 21 22 23
```

```
$ close(0,0)
```

```
$ close(0,1)
```

```
$ open(2118, "file2.c", 03)
```

```
the file file2.c fd:0
```

```
$ write(0, buf, 1636)
```

```
$ close(0,0 )
```

```
$ dir
```

```
CURRENT DIRECORY:
```

```
.          d xxx xxx xxx <dir>block chain,1
..         d xxx xxx xxx <dir>block chain,2
subdir     d xxx xxx xxx <dir>block chain,11
file2.c    f xxx --- --- 3427    block chain, 17 18 19 20 24 25 26
file3.c    f xxx --- --- 4396    block chain, 4 5 6 7 8 9 10 21 22 23
```

```
$ chdir ..
```

```
$ logout (2118)
```

```
no user in the file system.
```

```
$ halt
```

```
the file system now is halt! good bye.
```

注意,本文件系统程序未使用命令交互解释工具 Shell,读者可从文程序中观察到这一点。

清华大学计算机系列教材

1. 计算机操作系统教程 (第2版)
2. 计算机操作系统教程 (第2版) 习题解答与实验指导
3. PASCAL程序设计 (第二版)
4. PASCAL程序设计习题与选解 (新编)
5. IBM PC汇编语言程序设计 (第2版)
6. IBM PC汇编语言程序设计例题习题集
7. IBM PC汇编语言程序设计实验教程
8. 计算机图形学 (新版)
9. 微型计算机技术及应用——从16位到32位
10. 微型计算机技术及应用——习题与实验题集
11. 微型计算机技术及应用——微型机软硬件开发指南
12. 计算机组成与结构 (第3版)
13. 计算机组成原理实验指导书与习题集
14. 计算机系统结构 (第二版)
15. 数据结构 (第二版)
16. 数据结构题集
17. 图论与代数结构
18. 数字逻辑与数字集成电路
19. 数字系统设计自动化
20. 计算机图形学基础
21. 编译原理
22. 数据结构 (用面向对象方法与C++描述)
23. 计算机网络与Internet教程
24. 多媒体技术基础
25. 多媒体技术基础实验指南
26. 数理逻辑与集合论 (第2版)
27. 数理逻辑与集合论 (第2版) 精要及题解

张尧学 等
张尧学
郑恩华
郑恩华
沈美明 等
温冬梅 等
沈美明
孙家广 等
戴梅萼
戴梅萼
戴梅萼
王爱英 等
王 诚 等
郑世民 等
严蔚敏 等
严蔚敏 等
戴一奇 等
王尔屹 等
薛宏熙 等
唐泽圣 等
吕映芝 等
殷人昆 等
张尧学 等
林建宗
谢青珩 等
石纯一 等
王 宏 等

ISBN 7-302-04004-4



9 787302 040040

定价: 11.00 元

责任编辑: 马莉洁 / 封面设计: 傅瑞华