

苏州大学实验报告

院、系	计算机学院	年级专业	19 计科图灵	姓名	张昊	学号	1927405160
课程名称	操作系统课程实践					成绩	
指导教师	李培峰	同组实验者	无	实验日期	2022 年 4 月 25 日		

实验名称 实验 5 Linux 文件系统设计

一. 实验目的

1. 熟悉 Linux 文件系统的文件和目录结构。
2. 掌握文件系统的基本特征。
3. 掌握常用的文件操作函数。
4. 理解文件存储空间的管理、文件的物理结构和目录结构以及文件操作的实现。
5. 加深对文件系统内部功能和实现过程的理解。
6. 深入理解 Linux 文件系统的原理。
7. 学习并理解 Linux 的 VFS 文件系统管理技术。
8. 学习并理解 Linux 的 ext2 文件系统实现技术。
9. 设计并实现一个简单的类 ext2 文件系统。

二. 实验内容

1. (实验 6.1: 文件备份实验)

编写 C 程序, 模拟实现 Linux 文件系统的简单 I/O 流操作: 备份文件, 将源文件 source.dat 备份为 target.dat 文件。实验要求如下:

- (1) 使用 C 语言库函数实现文件备份。
- (2) 使用系统调用函数实现文件备份。

2. (实验 6.2: 简单文件系统的模拟)

模拟实现一个简单的二级文件管理系统, 要求做到以下几点。

- (1) 可以实现常用文件目录和文件操作, 比如:

```
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) 在列文件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 11: 设计一个简单的文件系统)

设计并实现一个类似于 ext2 但能够对磁盘上的数据块进行加密的文件系统 myext2。本实验的主要内容如下。

- (1) 添加一个类似于 ext2 的文件系统 myext2。

- (2) 修改 myext2 文件系统的 magic number。
- (3) 修改文件系统操作。
- (4) 添加文件系统创建工具。

对于 myext2 文件系统，要求如下：

- (1) myext2 文件系统的物理格式定义与 ext2 文件系统基本一致，但 myext2 文件系统的 magic number 是 0x6666，而 ext2 文件系统的 magic number 是 0xEF53。
- (2) myext2 文件系统是 ext2 文件系统的定制版本，前者不但支持 ext2 文件系统的部分操作，而且添加了文件系统创建工具。

三. 操作方法和实验步骤

(一) 实验 6.1: 文件备份实验

1. 使用 C 语言库函数实现
 - (1) 使用 fopen()函数以只读方式打开想要备份的源文件 source，并以只写方式打开写入内容的目标文件 target。
 - (2) 使用 fread()函数循环读取源文件中一个缓冲区大小的内容，然后使用 fwrite()函数将读取的内容写入目标文件。
 - (3) 读取与写入完毕后，使用 fclose()函数关闭读写文件流。

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, const char **argv) {
    printf("backup-libc 基于 C 语言库函数实现的文件备份.\n");
    if (argc != 3) {
        fprintf(stderr, "需要两个参数: 原文件 目标文件\n");
        exit(EXIT_FAILURE);
    }
    //打开文件
    FILE *source, *target;
    source = fopen(argv[1], "r");
    target = fopen(argv[2], "w");
    if (source == NULL) {
        fprintf(stderr, "打开原文件失败.\n");
        exit(EXIT_FAILURE);
    }
    if (target == NULL) {
        fprintf(stderr, "创建备份文件失败.\n");
        exit(EXIT_FAILURE);
    }
    //备份文件
    char buf;
    while (fread(&buf, sizeof(buf), 1, source) == 1) {
        if (!fwrite(&buf, sizeof(buf), 1, target)) {
            fprintf(stderr, "写备份文件失败.\n");
        }
    }
}
```

```

        exit(EXIT_FAILURE);
    }
}
if (ferror(source) != 0) {
    fprintf(stderr, "读原文件失败.\n");
    exit(EXIT_FAILURE);
}
//关闭文件
if (fclose(source)) {
    fprintf(stderr, "关闭原文件失败.\n");
    exit(EXIT_FAILURE);
}
if (fclose(target)) {
    fprintf(stderr, "关闭备份文件失败.\n");
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}

```

2. 使用系统调用函数实现

- (1) 使用 `open()` 系统调用函数以只读方式打开想要备份的源文件 `source`，并以只写方式打开想要写入内容的目标文件 `target`。
- (2) 使用 `read()` 系统调用函数循环读取源文件中一个缓冲区大小的内容，然后使用 `write()` 系统调用函数将读取的内容写入目标文件。
- (3) 读取与写入完毕后，使用 `close()` 系统调用函数关闭读写文件流。

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define MAXSIZE 1024

int main(int argc, const char **argv) {
    printf("backup-syscall 基于 Linux 系统调用函数实现的文件备份.\n");
    if (argc != 3) {
        fprintf(stderr, "需要两个参数: 原文件 目标文件\n");
        exit(EXIT_FAILURE);
    }
    //打开文件
    int source, target;
    source = open(argv[1], O_RDONLY);
    target = open(argv[2], O_WRONLY | O_CREAT, 0644);
    if (source == -1) {
        fprintf(stderr, "打开原文件失败.\n");
        exit(EXIT_FAILURE);
    }
    if (target == -1) {

```

```

    fprintf(stderr, "创建备份文件失败.\n");
    exit(EXIT_FAILURE);
}
//备份文件
ssize_t size;
char buf[MAXSIZE];
while ((size = read(source, buf, MAXSIZE)) > 0) {
    if (write(target, buf, size) != size) {
        fprintf(stderr, "写备份文件失败.\n");
        exit(EXIT_FAILURE);
    }
}
if (size < 0) {
    fprintf(stderr, "读原文件失败.\n");
    exit(EXIT_FAILURE);
}
//关闭文件
if (close(source) < 0) {
    fprintf(stderr, "关闭原文件失败.\n");
    exit(EXIT_FAILURE);
}
if (close(target) < 0) {
    fprintf(stderr, "关闭备份文件失败.\n");
    exit(EXIT_FAILURE);
}
exit(EXIT_SUCCESS);
}

```

（二）实验 6.2: 简单文件系统的模拟

文件系统采用两级目录，第一级对应用户的家目录，第二级对应用户所属的文件。所有文件均为文本文件，支持多个用户的文件管理（最多 100 个用户），但一个用户只能管理属于自己的文件，不考虑文件共享；磁盘可以存储用户文件的总大小为 512KB。与教材样例代码不同的是，文件在磁盘中采用链接分配，512KB 的磁盘被划分为 1024 个块，每块 512 字节；每个文件创建时即分配一个磁盘块，后续根据写入量对文件分配的磁盘块数量进行扩充。

设计了如下数据结构实现该文件系统。

（1）磁盘块

```

typedef struct disk_block { //磁盘块结构体
    int address; //起始地址
    int length; //使用容量（最大为 BLOCK_SIZE）
    struct disk_block *next; //指向下一磁盘块的指针
} DISK;

```

多个磁盘块串联形成链表，用于抽象文件在磁盘中的存储。

（2）文件控制块（FCB）

```

typedef struct file_control_block { //文件控制块

```

```

char file_name[16]; //文件名
DISK *disk_start, *disk_end; //文件在磁盘中的起始磁盘块和结束磁盘块
int length; //文件内容长度
int max_length; //文件占用空间
char mode; //文件读写方式
time_t modify_time; //文件相关的时间信息
bool opened; //判断是否有进程打开了该文件
} FCB;

```

记录了文件的文件名，起始磁盘块和结束磁盘块，文件大小，占用空间，读写方式，文件修改时间以及打开情况。

(3) 用户文件目录 (user file directory, UFD)，即二级目录

```

typedef struct user_file_directory { //用户文件目录
    FCB *file; //文件
    struct user_file_directory* next; //下一个目录
} UFD;

```

用户目录下的文件使用链表来存储。

(4) 主文件目录 (master file directory, MFD)，即一级目录

```

typedef struct master_file_directory { //主文件目录
    char username[16]; //用户账号
    char password[16]; //用户密码
    UFD *user_dir_head; //用户文件目录 (二级目录)
} MFD;

```

记录了用户名和密码，其中用户名最长 15 位，最短 3 位；密码最短 6 位，最长 15 位。各个用户的家目录使用数组存储，其中包含了指向二级目录链表的头节点。

使用数组来模拟 512KB 的磁盘空间，并给出长度为 1024 的数组用于表示每个磁盘块的使用情况。记录用户数量和当前登录的用户 ID，便于文件的管理。

```

char memory[MAX_DISK_SIZE]; //512KB 的磁盘存储空间 (链接分配)
char memory_usage[BLOCK_AMOUNT] = {}; //磁盘块的使用情况
MFD users[MAX_USER_SIZE]; //一级目录
int user_count = 0; //用户数量
int login_user_id = -1; //目前登录的用户 ID

```

为磁盘块的分配和回收操作单独实现了函数，其中分配是逐个分配的，回收是回收整个磁盘块的链表：

```

//分配一个磁盘块
DISK *memory_alloc() {
    int address = -1;
    for (int i = 0; i < BLOCK_AMOUNT; ++i) {
        if (memory_usage[i] == 0) {
            address = i * BLOCK_SIZE;
            memory_usage[i] = 1;
            break;
        }
    }
    if (address == -1) {
        return NULL;
    }
}

```

```

    }
    DISK *block = (DISK *) malloc(sizeof(DISK));
    block->address = address;
    block->length = 0;
    block->next = NULL;
    return block;
}
//回收 start_block 指向的磁盘块链表
void memory_free(DISK *start_block) {
    DISK *next;
    while (start_block != NULL) {
        next = start_block->next;
        memory_usage[start_block->address / BLOCK_SIZE] = 0;
        free(start_block);
        start_block = next;
    }
}

```

实现了常用文件目录和文件操作,对于文件管理的规则大致是:同一个用户不能创建同名文件;用户只能对存在的文件进行删除、打开、关闭、读取、写入、重命名和修改文件模式;用户新建的文件默认是可读可写的,处于关闭状态,占用一个磁盘块,文件大小为 0;文件只能被一个进程打开一次,只有打开的进程才能读取和写入,且写入需要文件具有相应权限;文件只有被关闭时才能被删除,删除的同时回收分配的磁盘空间;当磁盘空间不足时,文件的新建和写入将受到影响。如下代码所示:

```

int file_create(const char *file_name) {
    //判断文件是否存在
    for (UFD *p = users[login_user_id].user_dir_head->next;
        p != NULL;
        p = p->next) {
        if (strcmp(p->file->file_name, file_name) == 0) {
            printf("文件%s 已存在\n", file_name);
            return 1;
        }
    }
    //创建目录项
    UFD *file_node = (UFD *) malloc(sizeof(UFD)); //分配 UFD
    if (file_node == NULL) {
        return 2;
    }
    file_node->next = NULL;
    //创建文件控制块
    file_node->file = (FCB *) malloc(sizeof(FCB)); //分配 FCB
    if (file_node->file == NULL) {
        return 2;
    }
    strcpy(file_node->file->file_name, file_name);
}

```

```

file_node->file->mode = FILE_MODE_WRITABLE;
file_node->file->disk_start = memory_alloc(); //分配一个磁盘块
if (file_node->file->disk_start == NULL) {
    printf("文件系统没有空余空间可用\n");
    free(file_node->file); //撤销分配
    free(file_node);
    return 3;
}
file_node->file->disk_end = file_node->file->disk_start; //结束磁盘块
file_node->file->length = 0; //文件大小
file_node->file->max_length = BLOCK_SIZE; //占用空间
file_node->file->opened = false; //默认未打开
file_node->file->modify_time = time(NULL); //文件修改时间
//将目录项插入 UFD 链表
UFD *p = users[login_user_id].user_dir_head;
while (p->next) p = p->next;
p->next = file_node;
return 0;
}

int file_delete(const char *file_name) {
    UFD *prev = users[login_user_id].user_dir_head; //前驱节点
    UFD *curr = prev->next; //要找的节点
    //查找指定的文件
    while (curr != NULL) {
        if (strcmp(curr->file->file_name, file_name) == 0) {
            break; //找到了
        } else {
            prev = prev->next;
            curr = curr->next;
        }
    }
    if (curr == NULL) {
        printf("没有找到该文件，请检查输入的文件名是否正确\n");
        return 1;
    }
    //判断文件是否被进程打开
    if (curr->file->opened == true) {
        printf("该文件已被进程打开\n");
        return 2;
    }
    //调整前驱节点的 next
    prev->next = curr->next;
    //删除 curr
    memory_free(curr->file->disk_start);
}

```

```

    free(curr);
    return 0;
}

//查找指定的FCB
FCB *find_file(const char *file_name) {
    UFD *prev = users[login_user_id].user_dir_head; //前驱节点
    UFD *curr = prev->next; //要找的节点
    //查找指定的文件
    while (curr != NULL) {
        if (strcmp(curr->file->file_name, file_name) == 0) {
            break; //找到了
        } else {
            prev = prev->next;
            curr = curr->next;
        }
    }
    if (curr == NULL) {
        return NULL;
    }
    return curr->file;
}

int file_open(const char *file_name) {
    //判断file_name文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s不存在\n", file_name);
        return 1;
    }
    //判断file_name文件是否打开
    if (fcb->opened == true) {
        printf("该文件已被进程打开\n");
        return 2;
    }
    fcb->opened = true;
    return 0;
}

int file_close(const char *file_name) {
    //判断file_name文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s不存在\n", file_name);
        return 1;
    }

```



```

    }
    fcb->opened = false;
    return 0;
}

int file_read(const char *file_name) {
    //判断 file_name 文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s 不存在\n", file_name);
        return 1;
    }
    //判断 file_name 文件是否打开
    if (fcb->opened != true) {
        printf("该文件没有被打开, 无法读写\n");
        return 2;
    }
    //输出文件内容, 每 50 字符换一次行
    int count = 0;
    for (DISK *block = fcb->disk_start; block != NULL; block = block->next)
    {
        for (int i = 0; i < block->length; ++i, ++count) {
            printf("%c", memory[block->address + i]);
            if (memory[block->address + i] == '\n') {
                count = 0;
            }
            if ((count + 1) % 50 == 0) {
                printf("\n");
            }
        }
    }
    return 0;
}

int file_write(const char *file_name) {
    //判断 file_name 文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s 不存在\n", file_name);
        return 1;
    }
    //判断 file_name 文件是否打开
    if (fcb->opened != true) {
        printf("该文件没有被打开, 无法读写\n");
        return 2;
    }

```

```

}
//判断 file_name 文件是否有写权限
if (fcb->mode != FILE_MODE_WRITABLE) {
    printf("该文件没有写权限，无法写入\n");
    return 3;
}
char buff[1030];
printf("请输入要写入的内容，以换行结束，不超过 1024 个字符：\n");
fflush(stdout);
fflush(stdin);
getchar();
scanf("%[^\\n]", buff);
//舍弃大于 1024 的部分，末尾加换行符
size_t len = strlen(buff);
len = len > 1024 ? 1024 : len;
buff[len] = '\\n';
buff[++len] = 0;
//修改写入时间
fcb->modify_time = time(NULL);
//写入文件
DISK *b = fcb->disk_end;
for (int i = 0; i < len; ++i) {
    if (b->length >= BLOCK_SIZE) { //当前磁盘块已满
        b->next = memory_alloc(); //申请新的磁盘块
        if (b->next == NULL) {
            printf("文件系统没有空余空间可用，第%d 个字符后的内容将被舍弃\\n", i);
            return 4;
        }
        b = b->next;
        fcb->max_length += BLOCK_SIZE; //更新占用空间
        fcb->disk_end = b;
    }
    //逐字符写入
    memory[b->address + b->length] = buff[i];
    b->length++;
    fcb->length++;
}
return 0;
}

int file_dir() {
    printf("文件名\\t 文件长度\\t 占用空间\\t 物理地址\\t 文件类型\\t 进程占用\\t 修改时间\\n");
    char time_str[128];
    for (UFD *p = users[login_user_id].user_dir_head->next;

```

```

        p != NULL;
        p = p->next) {
    strftime(time_str, 128, "%a, %d %b %Y %H:%M:%S GMT",
            localtime(&p->file->modify_time));
    printf("%-6s\t %-6d\t %-6d\t %-6d\t %-6c\t %-6s\t %s\n",
            p->file->file_name,
            p->file->length,
            p->file->max_length,
            p->file->disk_start->address,
            p->file->mode,
            p->file->opened == true ? "是" : "否",
            time_str);
    }
    return 0;
}

int file_rename(const char *file_name, const char *new_name) {
    //判断 file_name 文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s 不存在\n", file_name);
        return 1;
    }
    //判断 new_name 文件是否存在
    for (UFD *p = users[login_user_id].user_dir_head->next;
        p != NULL;
        p = p->next) {
        if (strcmp(p->file->file_name, new_name) == 0) {
            printf("文件%s 已存在\n", new_name);
            return 2;
        }
    }
    strcpy(fcb->file_name, new_name);
    return 0;
}

int file_chmod(const char *file_name, char mod) {
    //判断 file_name 文件是否存在
    FCB *fcb = find_file(file_name);
    if (fcb == NULL) {
        printf("文件%s 不存在\n", file_name);
        return 1;
    }
    //判断 file_name 文件是否打开
    if (fcb->opened == true) {

```

```

    printf("该文件已被进程打开\n");
    return 2;
}
//判断文件模式是否合法
if (mod != FILE_MODE_READONLY && mod != FILE_MODE_WRITABLE) {
    printf("错误的文件模式\n");
    return 3;
}
fcb->mode = mod;
return 0;
}

```

实现了基本的用户管理，包括：用户注册、登录以及登出。特别注意的是，用户输入密码的时候会关闭输入回显。代码如下：

```

//用户管理
void user_create() {
    if (user_count >= MAX_USER_SIZE) {
        printf("用户数量已达到最大，创建用户失败\n");
        return;
    }
    char input_cache[32], input_cache2[32];
    //输入并校验用户名
    printf("请输入用户名: ");
    fflush(stdout);
    fflush(stdin);
    scanf("%s", input_cache);
    fflush(stdin);
    if (strlen(input_cache) <= 2) {
        printf("用户名过短，创建用户失败\n");
        return;
    }
    if (strlen(input_cache) > 16) {
        input_cache[16] = 0;
        printf("用户名过长，将截取前 15 个字符: %s\n", input_cache);
    }
    for (int i = 0; i < user_count; i++) {
        if (!strcmp(users[i].username, input_cache)) {
            printf("该用户名已存在，创建用户失败\n");
            return;
        }
    }
    int user_id = user_count;
    user_count++;
    strcpy(users[user_id].username, input_cache);
    //输入并校验密码
    for (;;) {

```

```

printf("请输入密码: ");
fflush(stdout);
fflush(stdin);
system("stty -echo"); //关闭回显
system("stty -icanon");
scanf("%s", input_cache);
fflush(stdin);
system("stty icanon");
system("stty echo");
printf("\n");
if (strlen(input_cache) < 6 || strlen(input_cache) > 16) {
    printf("密码长度应为 6-15 位, 请重新输入\n");
    continue;
}
printf("请重复输入: ");
fflush(stdout);
fflush(stdin);
system("stty -echo"); //关闭回显
system("stty -icanon");
scanf("%s", input_cache2);
fflush(stdin);
system("stty icanon");
system("stty echo");
printf("\n");
if (strcmp(input_cache, input_cache2) != 0) {
    printf("两次输入的不一致, 请重新输入\n");
    continue;
}
break;
}
strcpy(users[user_id].password, input_cache);
printf("创建用户成功, 将以 %s 用户登录\n", users[user_id].username);
login_user_id = user_id;
//创建用户的二级目录
users[user_id].user_dir_head = (UFD *) malloc(sizeof(UFD)); //头节点!
users[user_id].user_dir_head->file = NULL;
}

void user_login() {
    char name[32], passwd[32];
    //输入并校验用户名
    printf("请输入用户名: ");
    fflush(stdout);
    fflush(stdin);
    scanf("%s", name);
    fflush(stdin);

```

```

if (strlen(name) <= 2) {
    printf("用户名过短\n");
    return;
}
if (strlen(name) > 16) {
    name[16] = 0;
    printf("用户名过长, 将截取前 15 个字符: %s\n", name);
}
int user_id = -1;
for (int i = 0; i < user_count; i++) {
    if (strcmp(users[i].username, name) == 0) {
        user_id = i;
        break;
    }
}
if (user_id == -1) {
    printf("您输入的用户名不存在\n");
    return;
}
//输入并校验密码
for (int times = 3; times > 0; times--) {
    printf("请输入密码: ");
    fflush(stdout);
    fflush(stdin);
    system("stty -echo"); //关闭回显
    system("stty -icanon");
    scanf("%s", passwd);
    fflush(stdin);
    system("stty icanon");
    system("stty echo");
    printf("\n");
    if (strcmp(users[user_id].password, passwd) != 0) {
        printf("您输入的密码错误, 您还有%d次输入机会\n", times - 1);
        continue;
    }
    printf("您好, %s\n", users[user_id].username);
    login_user_id = user_id;
    break;
}
}

void user_logout() {
    printf("再见, %s!\n", users[login_user_id].username);
    login_user_id = -1;
}

```

主函数使用菜单驱动的方法, 由用户输入命令, 程序执行并返回结果, 其中程序正确执行不输

出，而是在出现错误时输出提示信息。主函数如下：

```
int main() {
    printf("文件管理模拟器\n");
    char input_cache[128], *input_res;
    for (;;) {
        printf("请登录或创建用户\n");
        printf("1-登录; 2-创建用户; 3-退出\n");
        printf(">>> ");
        fflush(stdout);
        scanf("%s", input_cache);
        fflush(stdin);
        int choice = (int) strtol(input_cache, &input_res, 10);
        if (choice == 1) {
            user_login();
        } else if (choice == 2) {
            user_create();
        } else if (choice == 3) {
            exit(EXIT_SUCCESS);
        } else {
            printf("您的输入有误，请重新选择\n");
        }
        fflush(stdout);
        if (login_user_id == -1) { //没有登录状态
            continue;
        }
        //主循环
        for (;;) {
            printf(">>> ");
            fflush(stdout);
            memset(input_cache, 0, sizeof(input_cache));
            getchar(); //注意 scanf 不会对换行符做特殊处理，需要把\n 单独去掉
            scanf("%s", input_cache); //使用"%s"代替 gets
            fflush(stdin);
            char *command = strtok(input_cache, " ");
            if (command == NULL || strlen(command) == 0) {
                continue;
            }
            if (strcmp(command, "create") == 0) {
                char *file_name = strtok(NULL, " ");
                if (file_name == NULL) {
                    printf("请输入文件名\n");
                } else {
                    if (file_create(file_name) != 0) {
                        printf("文件创建失败\n");
                    }
                }
            }
        }
    }
}
```

```

    }
} else if (strcmp(command, "delete") == 0) {
    char *file_name = strtok(NULL, " ");
    if (file_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_delete(file_name) != 0) {
            printf("文件删除失败\n");
        }
    }
} else if (strcmp(command, "open") == 0) {
    char *file_name = strtok(NULL, " ");
    if (file_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_open(file_name) != 0) {
            printf("文件打开失败\n");
        }
    }
} else if (strcmp(command, "close") == 0) {
    char *file_name = strtok(NULL, " ");
    if (file_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_close(file_name) != 0) {
            printf("文件关闭失败\n");
        }
    }
} else if (strcmp(command, "read") == 0) {
    char *file_name = strtok(NULL, " ");
    if (file_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_read(file_name) != 0) {
            printf("文件读取失败\n");
        }
    }
} else if (strcmp(command, "write") == 0) {
    char *file_name = strtok(NULL, " ");
    if (file_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_write(file_name) != 0) {
            printf("文件写入失败\n");
        }
    }
}

```



```

    }
} else if (strcmp(command, "dir") == 0) {
    file_dir();
} else if (strcmp(command, "rename") == 0) {
    char *file_name = strtok(NULL, " ");
    char *new_name = strtok(NULL, " ");
    if (file_name == NULL || new_name == NULL) {
        printf("请输入文件名\n");
    } else {
        if (file_rename(file_name, new_name) != 0) {
            printf("修改名称失败\n");
        }
    }
} else if (strcmp(command, "chmod") == 0) {
    char *file_name = strtok(NULL, " ");
    char *type_str = strtok(NULL, " ");
    if (file_name == NULL || type_str == NULL) {
        printf("请输入文件名以及文件模式\n");
    } else {
        char mod = type_str[0];
        if (file_chmod(file_name, mod) != 0) {
            printf("文件模式修改失败\n");
        }
    }
} else if (strcmp(command, "help") == 0) {
    help();
} else if (strcmp(command, "logout") == 0) {
    user_logout();
    break;
} else if (strcmp(command, "exit") == 0) {
    exit(EXIT_SUCCESS);
} else {
    printf("您的输入有误, 请重新输入。 \n 输入 help 查看帮助。 \n");
}
fflush(stdout);
}
}
}

```

(三) 实验 11: 设计一个简单的文件系统

下面的操作步骤以在 4.15.0 版本的 Linux 内核上运行。

1. 添加一个类似于 ext2 的文件系统 myext2

通过查看 Linux 内核源代码, 得知其中属于 ext2 文件系统的文件有:

```
fs/ext2/acl.c
```

```
fs/ext2/acl.h
fs/ext2/balloc.c
fs/ext2/dir.c
fs/ext2/ext2.h
fs/ext2/file.c
fs/ext2/ialloc.c
fs/ext2/inode.c
fs/ext2/ioctl.c
fs/ext2/Kconfig
fs/ext2/Makefile
fs/ext2/namei.c
fs/ext2/super.c
fs/ext2/symlink.c
fs/ext2/xattr.c
fs/ext2/xattr.h
fs/ext2/xattr_security.c
fs/ext2/xattr_trusted.c
fs/ext2/xattr_user.c
```

(1) 复制源代码

第一步是通过复制源代码，添加 **myext2** 文件系统的源代码到 Linux 源代码。具体操作为把 **ext2** 文件系统的源代码复制到 **myext2** 文件系统中。按照 Linux 源代码的组织结构，把 **myext2** 文件系统的源代码存放到 **fs/myext2** 下，头文件则存放到 **include/linux** 下。在 Linux Shell 下执行如下操作：

```
# cd /usr/src/linux-4.15    #内核源代码目录
# cd fs
# cp -R ext2 myext2
# cd /usr/src/linux-4.15/fs/myext2
# mv ext2.h myext2.h
# cd /lib/modules/$(uname -r)/build/include/linux
# cp ext2_fs.h myext2_fs.h
# cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops
# cp ext2-atomic.h myext2-atomic.h
# cp ext2-atomic-setbit.h myext2-atomic-setbit.h
```

(2) 修改文件的内容

为了使复制的源代码可以正确编译，接下来修改上面添加的文件的内容。

① 使用如下脚本将原来文件中的 **EXT2** 替换成 **MYEXT2**，**ext2** 替换成 **myext2**。

```
#!/bin/bash
SCRIPT=substitute.sh
for f in *; do
    if [ $f = $SCRIPT ]; then
        echo "skip $f"
        continue
    fi
    echo -n "substitute ext2 to myext2 in $f..."
    cat $f | sed 's/ext2/myext2/g' >${f}_tmp
    mv ${f}_tmp $f
```

```

echo "done"
echo -n "substitute EXT2 to MYEXT2 in $f..."
cat $f | sed 's/EXT2/MYEXT2/g' >${f}_tmp
mv ${f}_tmp $f
echo "done"
done

```

脚本命名为 `substitute.sh`，在 `fs/myext2` 目录下执行。注意执行时需要加上可执行权限 (x)，需删除目录中 *.o 文件。

② 将如下文件

```

/lib/modules/$(uname -r)/build/include/linux/myext2_fs.h,
/lib/modules/$(uname -r)/build/include/asm-generic/bitops/myext2-atomic.h
/lib/modules/$(uname -r)/build/include/asm-generic/bitops/myext2-atomic-setbit.h

```

中的 `ext2`、`EXT2` 分别替换成 `myext2`、`MYEXT2`。

③ 做如下修改：

在 `/lib/modules/$(uname -r)/build/include/asm-generic/bitops.h` 文件中添加：

```
#include <asm-generic/bitops/myext2-atomic.h>
```

在 `/lib/modules/$(uname -r)/build/arch/x86/include/asm/bitops.h` 文件中添加：

```
#include <asm-generic/bitops/myext2-atomic-setbit.h>
```

在 `/lib/modules/$(uname -r)/build/include/uapi/linux/magic.h` 文件中添加：

```
#define MYEXT2_SUPER_MAGIC 0xEF53
```

(3) 把 `myext2` 编译成内核模块

修改 `myext2/Makefile` 文件：

```

# Makefile for the linux myext2-filesystem routines
obj-m := myext2.o
myext2-y := balloc.o dir.o file.o ialloc.o inode.o \
    ioctl.o namei.o super.o symlink.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
    make -C $(KDIR) M=$(PWD) modules

```

`myext2` 目录下执行：

```

# make      # 编译内核模块
# insmod myext2.ko  # 加载文件系统
# cat /proc/filesystems |grep myext2  # 查看 myext2 文件系统是否加载成功

```

(4) 对 `myext2` 文件系统进行测试

```

# cd
# dd if=/dev/zero of=myfs bs=1M count=1
# /sbin/mkfs.ext2 myfs
# mount -t myext2 -o loop ./myfs /mnt
# mount
# umount /mnt      # 卸载文件系统
# mount -t ext2 -o loop ./myfs /mnt
# mount
# umount /mnt      # 卸载文件系统
# rmmod myext2     # 卸载模块

```

2. 修改 myext2 文件系统的 magic number

找到 myext2 文件系统的 magic number, 并将其值改为 0x6666。4.15.0 版本的内核中, 该值在 include/uapi/linux/magic.h, 修改为:

```
#define MYEXT2_SUPER_MAGIC 0x6666
```

修改后, 使用 make 命令重新编译内核模块, 然后使用 insmod 命令安装编译好的 myext2.ko 内核模块。

```
# make      # 编译内核模块
# insmod myext2.ko  # 加载文件系统
# cat /proc/filesystems |grep myext2  # 查看 myext2 文件系统是否加载成功
```

编写程序 changeMN.c 修改创建的 myfs 文件系统的 magic number, 使其与内核中记录的 myext2 文件系统的 magic number 相匹配:

```
#include <stdio.h>
int main(int argc, const char **argv) {
    int ret;
    FILE *fp_read;
    FILE *fp_write;
    unsigned char buf[2048];
    fp_read = fopen(argc >= 2 ? argv[1] : "./myfs", "rb");
    if (fp_read == NULL) {
        printf("open myfs failed!\n");
        return 1;
    }
    fp_write = fopen(argc >= 3 ? argv[2] : "./fs.new", "wb");
    if (fp_write == NULL) {
        printf("open fs.new failed!\n");
        return 2;
    }
    ret = fread(buf, sizeof(unsigned char), 2048, fp_read);
    printf("previous magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
    buf[0x438] = 0x66;
    buf[0x439] = 0x66;
    fwrite(buf, sizeof(unsigned char), 2048, fp_write);
    printf("current magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
    while (ret == 2048) {
        ret = fread(buf, sizeof(unsigned char), 2048, fp_read);
        fwrite(buf, sizeof(unsigned char), ret, fp_write);
    }
    if (ret < 2048 && feof(fp_read)) {
        printf("change magic number ok!\n");
    }
    fclose(fp_read);
    fclose(fp_write);
    return 0;
}
```

对 changeMN.c 进行编译后, 将产生名为 changeMN 的可执行程序:

```
# gcc -o changeMN changeMN.c
```

对修改 magic number 后的 myext2 文件系统进行测试:

```
# dd if=/dev/zero of=myfs bs=1M count=1
# /sbin/mkfs.ext2 myfs
# ./changeMN myfs
# mount -t myext2 -o loop ./fs.new /mnt
# mount |tail
# umount /mnt      # 卸载文件系统
# mount -t ext2 -o loop ./fs.new /mnt
# rmmmod myext2    # 卸载模块
```

3. 修改文件系统操作

参照 ext2 文件系统中 mknod 操作的实现 ext2_mknod(), 修改 myext2 文件系统下的 mknod 操作:

```
/* fs/myext2/namei.c */
static int myext2_mknod (struct inode * dir, struct dentry *dentry,
                        int mode, int rdev){
    printk(KERN_ERR "oops, `mknod' is not supported by myext2!\n");
    return -EPERM;
    //其他代码注释掉
}
```

上述代码指明 mknod 操作不被支持, 并告知 Shell, 将错误号为 EPERM 的结果返回。

使用 make 重新编译内核模块, 再使用 insmod 命令安装编译好的 myext2.ko 内核模块:

```
# make      # 编译内核模块
# insmod myext2.ko  # 加载文件系统
# cat /proc/filesystems |grep myext2  # 查看 myext2 文件系统是否加载成功
```

在 Shell 中执行如下测试:

```
# mount -t myext2 -o loop ./fs.new /mnt
# cd /mnt
# mknod myfifo p      # 创建一个名为 myfifo 的命名管道
# cd
# umount /mnt        # 卸载文件系统
```

注: 如不显示 printk 打印的信息, 可以通过命令 dmesg|tail 查看。

4. 添加文件系统创建工具

使用如下 Shell 脚本来创建一个 myext2 文件系统创建工具, 命名为 mkfs.myext2:

```
#!/bin/bash
/sbin/losetup -d /dev/loop2      # 卸载占用的/dev/loop2
/sbin/losetup /dev/loop2 $1      # 第一个参数代表的文件装到/dev/loop2
/sbin/mkfs.ext2 /dev/loop2      # 使用 mkfs.ext2 格式化/dev/loop2
dd if=/dev/loop2 of=./tmpfs bs=1k count=2    # 头 2KB 内容取到临时文件系统
./changeMN $1 ./tmpfs          # 调用程序 changeMN
dd if=./fs.new of=/dev/loop2    # 写回头 2KB 内容
/sbin/losetup -d /dev/loop2      # 卸载文件系统
rm -f ./tmpfs                  # 清理临时文件系统
```

脚本的输入是一个文件名, 其大小为 myext2 文件系统的大小; 脚本的输出则是带有 myext2 文件系统的文件。

最后进行如下测试:

```
# dd if=/dev/zero of=myfs bs=1M count=1
# chmod +x mkfs.myext2
# ./mkfs.myext2 myfs
# mount -t myext2 -o loop ./myfs /mnt
# mount | grep myext2
# umount /mnt      # 卸载文件系统
# rmdir myext2     # 卸载模块
```

四. 实验结果和分析

(一) 实验 6.1: 文件备份实验

这个实验在示例程序的基础上增加了命令行参数，使得可以备份任意的文件。
使用 C 语言库函数备份文件的运行结果：

```
holger-405160@hao-zhang:~/code/exp05/backup$ gcc backup-libc.c -o backup-libc
holger-405160@hao-zhang:~/code/exp05/backup$ ./backup-libc source.dat target.dat
backup-libc 基于C语言库函数实现的文件备份.
holger-405160@hao-zhang:~/code/exp05/backup$ ls -lh *.dat
-rw-r--r-- 1 holger-405160 holger-405160 9.3K  4月 25 11:11 source.dat
-rw-r--r-- 1 holger-405160 holger-405160 9.3K  4月 25 12:07 target.dat
holger-405160@hao-zhang:~/code/exp05/backup$
```

使用系统调用备份文件的运行结果：

```
holger-405160@hao-zhang:~/code/exp05/backup$ gcc backup-syscall.c -o backup-syscall
holger-405160@hao-zhang:~/code/exp05/backup$ ./backup-syscall source.dat target.dat
backup-syscall 基于Linux系统调用函数实现的文件备份.
holger-405160@hao-zhang:~/code/exp05/backup$ ls -lh *.dat
-rw-r--r-- 1 holger-405160 holger-405160 9.3K  4月 25 11:11 source.dat
-rw-r--r-- 1 holger-405160 holger-405160 9.3K  4月 25 12:08 target.dat
holger-405160@hao-zhang:~/code/exp05/backup$
```

(二) 实验 6.2: 简单文件系统的模拟

下面的运行结果中，创建了两个用户 user1 和 user2，并分别新建了几个文件，测试了打开、关闭、读取、写入等操作：

```
holger-405160@hao-zhang:~/code/exp05/filesystem$ gcc filesystem.h filesystem.c -o filesystem
holger-405160@hao-zhang:~/code/exp05/filesystem$ ./filesystem
文件管理模拟器
请登录或创建用户
1-登录; 2-创建用户; 3-退出
>>> 2
请输入用户名: user1
请输入密码:
请重复输入:
创建用户成功, 将以 user1 用户登录
>>> create file1
>>> create file2
>>> create file3
>>> dir
文件名    文件长度    占用空间    物理地址    文件类型    进程占用    修改时间
file1     0           512        0           w          否         Mon, 25 Apr 2022 12:24:09 GMT
file2     0           512        512         w          否         Mon, 25 Apr 2022 12:24:14 GMT
file3     0           512        1024        w          否         Mon, 25 Apr 2022 12:24:19 GMT
>>> open file1
>>> write file1
请输入要写入的内容, 以换行结束, 不超过1024个字符:
123456789 987654321
>>> read file1
123456789 987654321
```

```
>>> write file2
该文件没有被打开，无法读写
文件写入失败
>>> delete file2
>>> dir
文件名 文件长度 占用空间 物理地址 文件类型 进程占用 修改时间
file1 20 512 0 w 是 Mon, 25 Apr 2022 12:24:41 GMT
file3 0 512 1024 w 否 Mon, 25 Apr 2022 12:24:19 GMT
>>> chmod file3 r
>>> open file3
>>> write file3
该文件没有写权限，无法写入
文件写入失败
>>> close file3
>>> logout
再见，user1!
```

```
请登录或创建用户
1-登录；2-创建用户；3-退出
>>> 2
请输入用户名：user2
请输入密码：
请重复输入：
创建用户成功，将以 user2 用户登录
>>> dir
文件名 文件长度 占用空间 物理地址 文件类型 进程占用 修改时间
>>> create file1
>>> dir
文件名 文件长度 占用空间 物理地址 文件类型 进程占用 修改时间
file1 0 512 512 w 否 Mon, 25 Apr 2022 12:26:43 GMT
>>> read file1
该文件没有被打开，无法读写
文件读取失败
>>> open file1
>>> read file1
>>> write file1
请输入要写入的内容，以换行结束，不超过1024个字符：
000111222 aaabbbccc
>>> write file1
请输入要写入的内容，以换行结束，不超过1024个字符：
**-- \\
>>> read file1
000111222 aaabbbccc
**-- \\
>>> logout
再见，user2!
请登录或创建用户
1-登录；2-创建用户；3-退出
>>> 3
holger-405160@hao-zhang:~/code/exp05/filesystem$ |
```

（三）实验 11：设计一个简单的文件系统

1. 添加一个类似于 ext2 的文件系统 myext2

（1）复制源代码

在 Linux Shell 下执行如下操作：

```
# cd /usr/src/linux-source-4.15.0 # 内核源代码目录
# cd fs
# cp -R ext2 myext2
# cd /usr/src/linux-source-4.15.0/fs/myext2
# mv ext2.h myext2.h
# cd /lib/modules/$(uname -r)/build/include/linux
# cp ext2_fs.h myext2_fs.h
# cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops
# cp ext2-atomic.h myext2-atomic.h
```

```
# cp ext2-atomic-setbit.h myext2-atomic-setbit.h
```

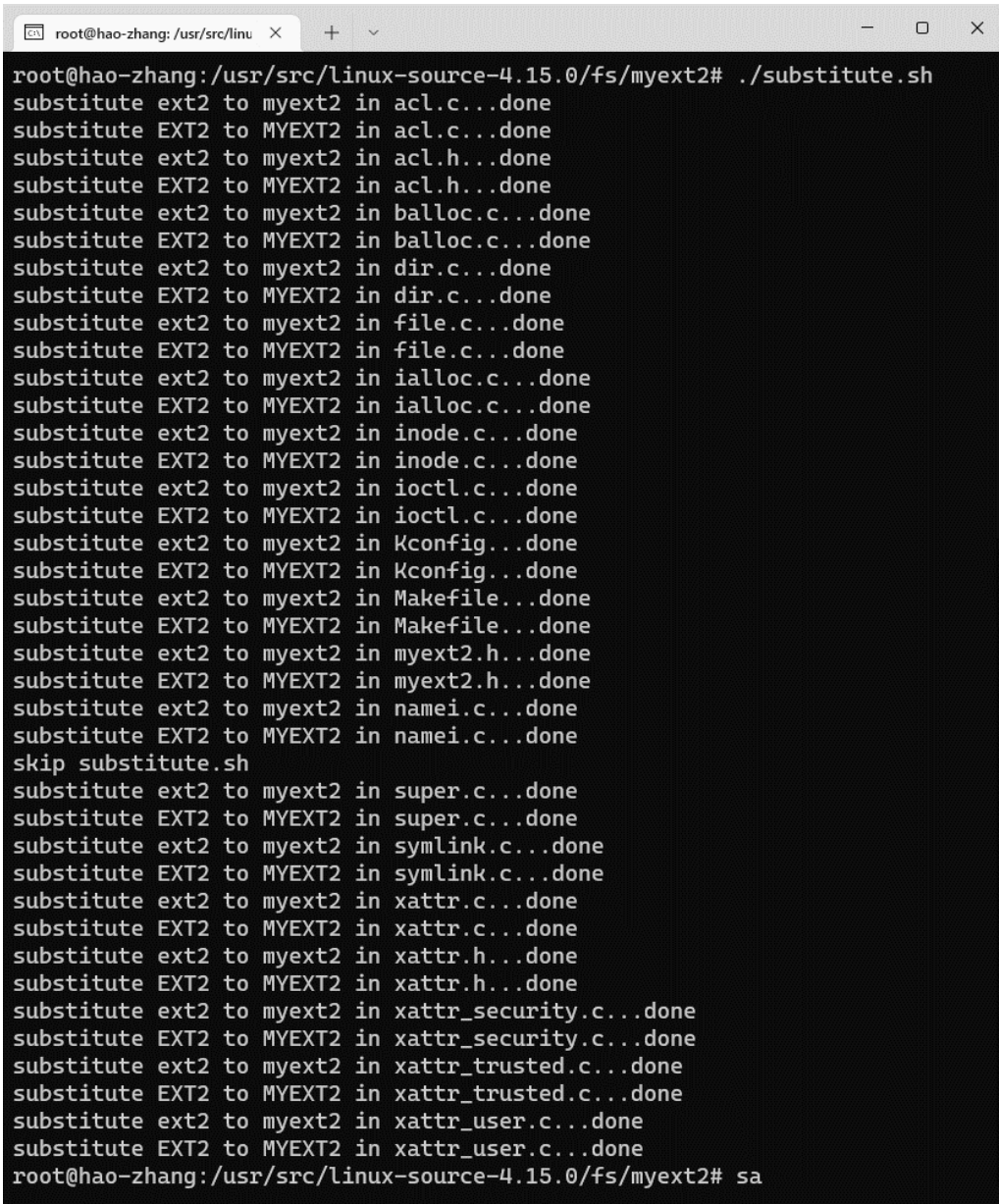
执行正确无输出结果。

```
root@hao-zhang:~# cd /usr/src/linux-source-4.15.0
root@hao-zhang:/usr/src/linux-source-4.15.0# cd fs
root@hao-zhang:/usr/src/linux-source-4.15.0/fs# cp -R ext2 myext2
root@hao-zhang:/usr/src/linux-source-4.15.0/fs# cd /usr/src/linux-source-4.15.0/fs/myext2
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# mv ext2.h myext2.h
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cd /lib/modules/$(uname -r)/build/include/linux
root@hao-zhang:/lib/modules/4.15.0-112-generic/build/include/linux# cp ext2_fs.h myext2_fs.h
root@hao-zhang:/lib/modules/4.15.0-112-generic/build/include/linux# cd /lib/modules/$(uname -r)/build/include/asm-generic
c/bitops
root@hao-zhang:/lib/modules/4.15.0-112-generic/build/include/asm-generic/bitops# cp ext2-atomic.h myext2-atomic.h
root@hao-zhang:/lib/modules/4.15.0-112-generic/build/include/asm-generic/bitops# cp ext2-atomic-setbit.h myext2-atomic-s
etbit.h
root@hao-zhang:/lib/modules/4.15.0-112-generic/build/include/asm-generic/bitops#
```

(2) 修改文件的内容

```
# cd /usr/src/linux-source-4.15.0/fs/myext2 # 进入 fs/myext2 目录
# vim substitute.sh # 编辑脚本
# chmod +x substitute.sh # 赋予可执行权限
# ./substitute.sh # 运行脚本
```

运行结果:



```
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# ./substitute.sh
substitute ext2 to myext2 in acl.c...done
substitute EXT2 to MYEXT2 in acl.c...done
substitute ext2 to myext2 in acl.h...done
substitute EXT2 to MYEXT2 in acl.h...done
substitute ext2 to myext2 in balloc.c...done
substitute EXT2 to MYEXT2 in balloc.c...done
substitute ext2 to myext2 in dir.c...done
substitute EXT2 to MYEXT2 in dir.c...done
substitute ext2 to myext2 in file.c...done
substitute EXT2 to MYEXT2 in file.c...done
substitute ext2 to myext2 in ialloc.c...done
substitute EXT2 to MYEXT2 in ialloc.c...done
substitute ext2 to myext2 in inode.c...done
substitute EXT2 to MYEXT2 in inode.c...done
substitute ext2 to myext2 in ioctl.c...done
substitute EXT2 to MYEXT2 in ioctl.c...done
substitute ext2 to myext2 in kconfig...done
substitute EXT2 to MYEXT2 in kconfig...done
substitute ext2 to myext2 in Makefile...done
substitute EXT2 to MYEXT2 in Makefile...done
substitute ext2 to myext2 in myext2.h...done
substitute EXT2 to MYEXT2 in myext2.h...done
substitute ext2 to myext2 in namei.c...done
substitute EXT2 to MYEXT2 in namei.c...done
skip substitute.sh
substitute ext2 to myext2 in super.c...done
substitute EXT2 to MYEXT2 in super.c...done
substitute ext2 to myext2 in symlink.c...done
substitute EXT2 to MYEXT2 in symlink.c...done
substitute ext2 to myext2 in xattr.c...done
substitute EXT2 to MYEXT2 in xattr.c...done
substitute ext2 to myext2 in xattr.h...done
substitute EXT2 to MYEXT2 in xattr.h...done
substitute ext2 to myext2 in xattr_security.c...done
substitute EXT2 to MYEXT2 in xattr_security.c...done
substitute ext2 to myext2 in xattr_trusted.c...done
substitute EXT2 to MYEXT2 in xattr_trusted.c...done
substitute ext2 to myext2 in xattr_user.c...done
substitute EXT2 to MYEXT2 in xattr_user.c...done
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# sa
```


分别使用 Vim 打开

/lib/modules/\$(uname -r)/build/include/linux/myext2_fs.h,

/lib/modules/\$(uname -r)/build/include/asm-generic/bitops/myext2-atomic.h

/lib/modules/\$(uname -r)/build/include/asm-generic/bitops/myext2-atomic-setbit.h

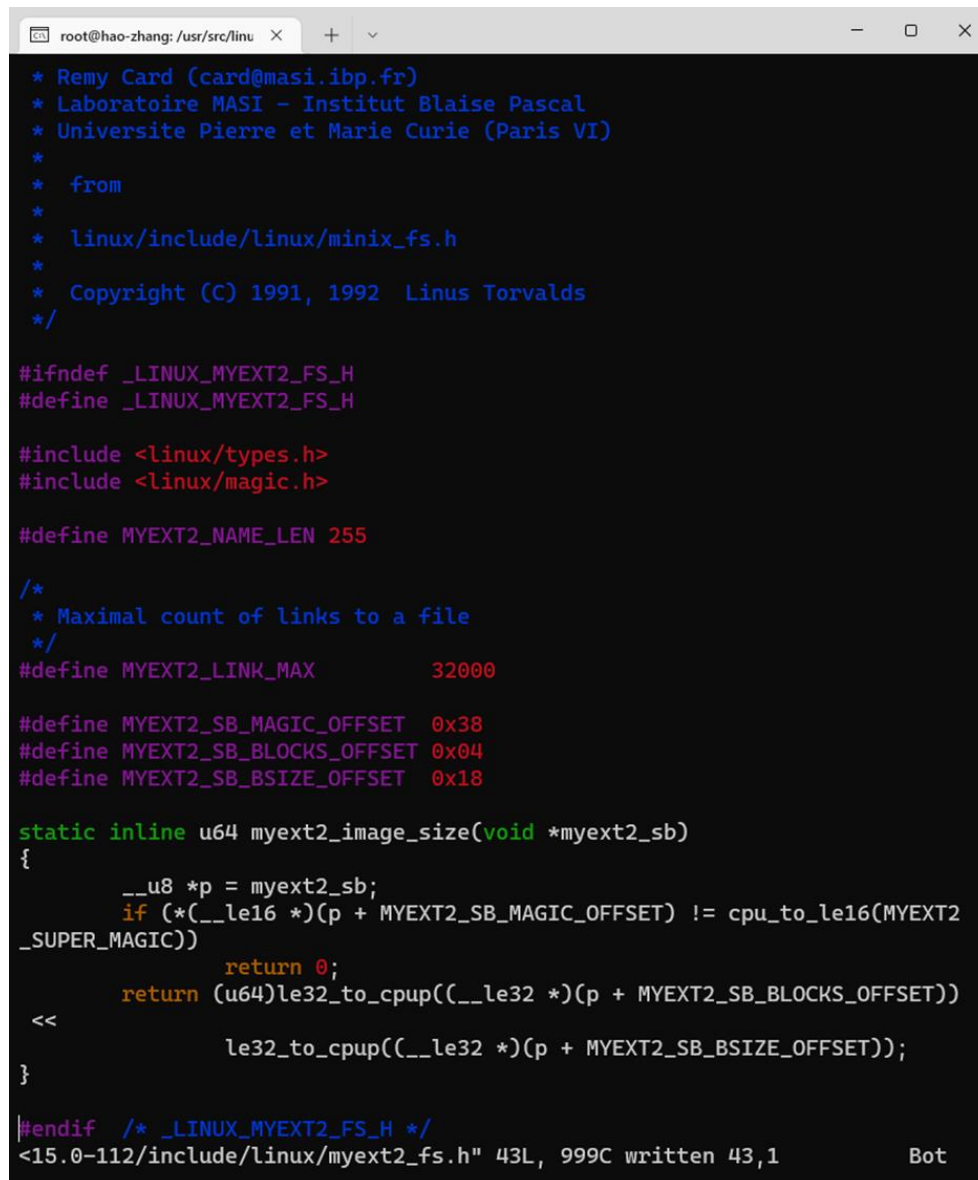
三个文件，依次输入

```
gg:., $s/ext2/myext2/g (回车)
```

```
gg:., $s/EXT2/MYEXT2/g (回车)
```

进行字符串替换。

如下图所示：



```
root@hao-zhang: /usr/src/linu x + v
* Remy Card (card@masi.ibp.fr)
* Laboratoire MASI - Institut Blaise Pascal
* Universite Pierre et Marie Curie (Paris VI)
*
* from
*
* linux/include/linux/minix_fs.h
*
* Copyright (C) 1991, 1992 Linus Torvalds
*/

#ifndef _LINUX_MYEXT2_FS_H
#define _LINUX_MYEXT2_FS_H

#include <linux/types.h>
#include <linux/magic.h>

#define MYEXT2_NAME_LEN 255

/*
 * Maximal count of links to a file
 */
#define MYEXT2_LINK_MAX 32000

#define MYEXT2_SB_MAGIC_OFFSET 0x38
#define MYEXT2_SB_BLOCKS_OFFSET 0x04
#define MYEXT2_SB_BSIZE_OFFSET 0x18

static inline u64 myext2_image_size(void *myext2_sb)
{
    __u8 *p = myext2_sb;
    if ((__le16 *) (p + MYEXT2_SB_MAGIC_OFFSET)) != cpu_to_le16(MYEXT2
_SUPER_MAGIC))
        return 0;
    return (u64) le32_to_cpup((__le32 *) (p + MYEXT2_SB_BLOCKS_OFFSET))
<<
        le32_to_cpup((__le32 *) (p + MYEXT2_SB_BSIZE_OFFSET));
}

#endif /* _LINUX_MYEXT2_FS_H */
<15.0-112/include/linux/myext2_fs.h" 43L, 999C written 43,1 Bot
```

使用 Vim 修改如下文件：

在 /lib/modules/\$(uname -r)/build/include/asm-generic/bitops.h 添加：

```
#include <asm-generic/bitops/myext2-atomic.h>
```

在 /lib/modules/\$(uname -r)/build/arch/x86/include/asm/bitops.h 添加：

```
#include <asm-generic/bitops/myext2-atomic-setbit.h>
```

在 /lib/modules/\$(uname -r)/build/include/uapi/linux/magic.h 添加：

```
#define MYEXT2_SUPER_MAGIC 0xEF53
```

如下图所示:

```
root@hao-zhang: ~  
#include <asm-generic/bitops/find.h>  
  
#ifndef _LINUX_BITOPS_H  
#error only <linux/bitops.h> can be included directly  
#endif  
  
#include <asm-generic/bitops/sched.h>  
#include <asm-generic/bitops/ffs.h>  
#include <asm-generic/bitops/hweight.h>  
#include <asm-generic/bitops/lock.h>  
  
#include <asm-generic/bitops/atomic.h>  
#include <asm-generic/bitops/non-atomic.h>  
#include <asm-generic/bitops/le.h>  
#include <asm-generic/bitops/ext2-atomic.h>  
#include <asm-generic/bitops/myext2-atomic.h>  
  
<2/include/asm-generic/bitops.h" 39L, 1165C written 37,45 95%
```

(3) 把 myext2 编译成内核模块

myext2 目录下执行:

```
# make      # 编译内核模块  
# insmod myext2.ko    # 加载文件系统  
# cat /proc/filesystems |grep myext2    # 查看 myext2 文件系统是否加载成功
```

```
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# vim Makefile  
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# make  
make -C /lib/modules/4.15.0-112-generic/build M=/usr/src/linux-source-4.15.0/fs/myext2 modules  
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-112-generic'  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/balloc.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/dir.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/file.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/ialloc.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/inode.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/ioctl.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/namei.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/super.o  
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/symlink.o  
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.o  
Building modules, stage 2.  
MODPOST 1 modules  
CC /usr/src/linux-source-4.15.0/fs/myext2/myext2.mod.o  
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.ko  
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-112-generic'  
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# insmod myext2.ko  
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cat /proc/filesystems |grep myext2  
myext2  
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# |
```

(4) 对 myext2 文件系统进行测试

确认 myext2 文件系统加载成功后对加载的 myext2 文件系统进行测试。进入家目录, 使用/dev/zero 文件创建了 myfs 文件, 并使用 ext2 文件系统对 myfs 文件进行了格式化:

```
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cd  
root@hao-zhang:~# dd if=/dev/zero of=myfs bs=1M count=1  
1+0 records in  
1+0 records out  
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00164778 s, 636 MB/s  
root@hao-zhang:~# /sbin/mkfs.ext2 myfs  
mke2fs 1.42.13 (17-May-2015)  
Discarding device blocks: done  
Creating filesystem with 1024 1k blocks and 128 inodes  
  
Allocating group tables: done  
Writing inode tables: done  
Writing superblocks and filesystem accounting information: done  
root@hao-zhang:~# |
```

使用 myext2 文件系统将文件 myfs 挂载到设备/mnt 上并查看：

```
root@hao-zhang:~# mount -t myext2 -o loop ./myfs /mnt
root@hao-zhang:~# mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1719388k,nr_inodes=429847,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=349984k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=28,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=21704)
configfs on /sys/kernel/config type configfs (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
vmhgfs-fuse on /mnt/hgfs type fuse.vmhgfs-fuse (rw,nosuid,nodev,relatime,user_id=0,group_id=0,allow_other)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_permissions,allow_other)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=349984k,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
/root/myfs on /mnt type myext2 (rw,relatime,errors=continue)
root@hao-zhang:~#
```

将文件 myfs 从/mnt 设备上卸载，并使用 ext2 文件系统重新挂载，操作也可以成功（因为没有做任何修改，和 ext2 文件系统是一样的）：

```
root@hao-zhang:~# umount /mnt
root@hao-zhang:~# mount -t ext2 -o loop ./myfs /mnt
root@hao-zhang:~# mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=1719388k,nr_inodes=429847,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=349984k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
tmpfs on /run/lock type tmpfs (rw,nosuid,nodev,noexec,relatime,size=5120k)
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup (rw,nosuid,nodev,noexec,relatime,xattr,release_agent=/lib/systemd/systemd-cgroups-agent,name=systemd)
pstore on /sys/fs/pstore type pstore (rw,nosuid,nodev,noexec,relatime)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/net_cls,net_prio type cgroup (rw,nosuid,nodev,noexec,relatime,net_cls,net_prio)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/rdma type cgroup (rw,nosuid,nodev,noexec,relatime,rdma)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=28,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=21704)
configfs on /sys/kernel/config type configfs (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
vmhgfs-fuse on /mnt/hgfs type fuse.vmhgfs-fuse (rw,nosuid,nodev,relatime,user_id=0,group_id=0,allow_other)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,nodev,relatime,user_id=0,group_id=0,default_permissions,allow_other)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=349984k,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
/root/myfs on /mnt type ext2 (rw,relatime)
root@hao-zhang:~#
```

卸载 myfs 文件，同时卸载刚才挂载的模块：

```
root@hao-zhang:~# umount /mnt
root@hao-zhang:~# rmmod myext2
root@hao-zhang:~#
```

2. 修改 myext2 文件系统的 magic number

修改 include/uapi/linux/magic.h:

```
root@hao-zhang: /usr/src/linux
#define CRAMFS_MAGIC_WEND      0x453dcd28      /* magic number with t
he wrong endianness */
#define DEBUGFS_MAGIC         0x64626720
#define SECURITYFS_MAGIC      0x73636673
#define SELINUX_MAGIC         0xf97cff8c
#define SMACK_MAGIC           0x43415d53      /* "SMAC" */
#define RAMFS_MAGIC           0x858458f6      /* some random number
*/
#define TMPFS_MAGIC           0x01021994
#define HUGETLBFS_MAGIC       0x958458f6      /* some random number
*/
#define SQUASHFS_MAGIC         0x73717368
#define ECRYPTFS_SUPER_MAGIC   0xf15f
#define EFS_SUPER_MAGIC       0x414A53
#define EXT2_SUPER_MAGIC      0xEF53
#define MYEXT2_SUPER_MAGIC    0x6666
#define EXT3_SUPER_MAGIC      0xEF53
#define XENFS_SUPER_MAGIC     0xabba1974
#define EXT4_SUPER_MAGIC      0xEF53
#define BTRFS_SUPER_MAGIC     0x9123683E
#define NILFS_SUPER_MAGIC     0x3434
#define F2FS_SUPER_MAGIC      0xF2F52010
#define HPFS_SUPER_MAGIC      0xf995e849
#include/uapi/linux/magic.h" 94L, 3436C written    23,28-32    13%
```

重新编译内核模块并安装:

```
root@hao-zhang:/usr/src/linux-source-4.15.0# cd fs/myext2/
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# make
make -C /lib/modules/4.15.0-112-generic/build M=/usr/src/linux-source-4.1
5.0/fs/myext2 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-112-generic'
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/balloc.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/dir.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/file.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/ialloc.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/inode.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/ioctl.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/namei.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/super.o
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/symlink.o
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-112-generic'
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# insmod myext2.ko
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cat /proc/filesyst
ems |grep myext2
myext2
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2#
```

编写程序 changeMN.c 并编译。对修改 magic number 后的 myext2 文件系统进行测试:

```

root@hao-zhang:~# vim changeMN.c
root@hao-zhang:~# gcc -o changeMN changeMN.c
root@hao-zhang:~# dd if=/dev/zero of=myfs bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00336777 s, 311 MB/s
root@hao-zhang:~# /sbin/mkfs.ext2 myfs
mke2fs 1.42.13 (17-May-2015)
Discarding device blocks: done
Creating filesystem with 1024 1k blocks and 128 inodes

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

root@hao-zhang:~# ./changeMN myfs
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
root@hao-zhang:~# mount -t myext2 -o loop ./fs.new /mnt
root@hao-zhang:~# mount | tail
configfs on /sys/kernel/config type configfs (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime,pagesize=2M)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
vmhgfs-fuse on /mnt/hgfs type fuse.vmhgfs-fuse (rw,nosuid,nodev,relatime,
user_id=0,group_id=0,allow_other)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,n
odev,relatime,user_id=0,group_id=0,default_permissions,allow_other)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=349984k
,mode=700,uid=1000,gid=1000)
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,r
elative,user_id=1000,group_id=1000)
/root/fs.new on /mnt type myext2 (rw,relatime,errors=continue)
root@hao-zhang:~#

```

成功挂载。卸载后，尝试使用 ext2 文件系统挂载，发现挂载失败：

```

root@hao-zhang:~# umount /mnt
root@hao-zhang:~# mount -t ext2 -o loop ./fs.new /mnt
mount: wrong fs type, bad option, bad superblock on /dev/loop0,
       missing codepage or helper program, or other error

       In some cases useful info is found in syslog - try
       dmesg | tail or so.
root@hao-zhang:~# rmmod myext2
root@hao-zhang:~# |

```

这是因为修改了 magic number 的 fs.new 文件无法与 ext2 文件系统匹配。

3. 修改文件系统操作

修改 myext2 文件系统中的 mknod 操作：

```
root@hao-zhang: /usr/src/linu x + v
mark_inode_dirty(inode);
d_tmpfile(dentry, inode);
unlock_new_inode(inode);
return 0;
}

static int myext2_mknod (struct inode * dir, struct dentry *dentry, umode
_t mode, dev_t rdev)
{
    printk(KERN_ERR "oops, 'mknod' is not supported by myext2!\n");
    return -EPERM;
}

// struct inode * inode;
// int err;
//
// err = dquot_initialize(dir);
// if (err)
//     return err;
//
// inode = myext2_new_inode (dir, mode, &dentry->d_name);
// err = PTR_ERR(inode);
// if (!IS_ERR(inode)) {
//     init_special_inode(inode, inode->i_mode, rdev);
// #ifdef CONFIG_MYEXT2_FS_XATTR
//     inode->i_op = &myext2_special_inode_operations;
// #endif
//     mark_inode_dirty(inode);
//     err = myext2_add_nondir(dentry, inode);
// }
// return err;
}

static int myext2_symlink (struct inode * dir, struct dentry * dentry,
const char * symname)
{
    struct super_block * sb = dir->i_sb;
    int err = -ENAMETOOLONG;
    unsigned l = strlen(symname)+1;
    struct inode * inode;

"namei.c" 452L, 10400C written          145,0-1      32%
```

重新编译内核模块并安装:

```
root@hao-zhang:~# cd /usr/src/linux-source-4.15.0/fs/myext2/
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# vim namei.c
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# make
make -C /lib/modules/4.15.0-112-generic/build M=/usr/src/linux-source-4.1
5.0/fs/myext2 modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-112-generic'
CC [M] /usr/src/linux-source-4.15.0/fs/myext2/namei.o
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
CC /usr/src/linux-source-4.15.0/fs/myext2/myext2.mod.o
LD [M] /usr/src/linux-source-4.15.0/fs/myext2/myext2.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-112-generic'
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# insmod myext2.ko
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cat /proc/filesyst
ems |grep myext2
myext2
root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# |
```

在 Shell 中执行测试:


```

root@hao-zhang:/usr/src/linux-source-4.15.0/fs/myext2# cd
root@hao-zhang:~# mount -t myext2 -o loop ./fs.new /mnt
root@hao-zhang:~# cd /mnt
root@hao-zhang:/mnt# mknod myfifo p
mknod: myfifo: Operation not permitted
root@hao-zhang:/mnt# dmesg|tail
[ 3695.729334] perf: interrupt took too long (7450 > 6835), lowering kern
el.perf_event_max_sample_rate to 26750
[ 4433.150999] perf: interrupt took too long (9764 > 9312), lowering kern
el.perf_event_max_sample_rate to 20250
[ 6215.735294] perf: interrupt took too long (12647 > 12205), lowering ke
rnel.perf_event_max_sample_rate to 15750
[ 6288.245485] perf: interrupt took too long (16209 > 15808), lowering ke
rnel.perf_event_max_sample_rate to 12250
[ 7353.066672] myext2: loading out-of-tree module taints kernel.
[ 7353.077563] myext2: module verification failed: signature and/or requi
red key missing - tainting kernel
[ 7830.757537] EXT4-fs (loop0): mounting ext2 file system using the ext4
subsystem
[ 7830.765929] EXT4-fs (loop0): mounted filesystem without journal. Opts:
(null)
[11557.980082] EXT4-fs (loop0): VFS: Can't find ext4 filesystem
[12100.846126] oops, 'mknod' is not supported by myext2!
root@hao-zhang:/mnt# |

```

可以看到 mknod 操作不被支持，并且可以通过命令 dmesg|tail 看到 printk 打印的信息。

4. 添加文件系统创建工具

创建 myext2 文件系统创建工具 mkfs.myext2 并进行测试：

```

root@hao-zhang:~# vim mkfs.myext2
root@hao-zhang:~# dd if=/dev/zero of=myfs bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00340753 s, 308 MB/s
root@hao-zhang:~# chmod +x mkfs.myext2
root@hao-zhang:~# ./mkfs.myext2 myfs
losetup: /dev/loop2: detach failed: No such device or address
mke2fs 1.42.13 (17-May-2015)
Discarding device blocks: done
Creating filesystem with 1024 1k blocks and 128 inodes

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

2+0 records in
2+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.0249818 s, 82.0 kB/s
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
2048+0 records in
2048+0 records out
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.0958739 s, 10.9 MB/s
root@hao-zhang:~# mount -t myext2 -o loop ./myfs /mnt
root@hao-zhang:~# mount | grep myext2
/root/myfs on /mnt type myext2 (rw,relatime,errors=continue)
root@hao-zhang:~#

```

可见挂载情况如图。最后卸载文件系统并卸载内核模块：

```

root@hao-zhang:~# umount /mnt
root@hao-zhang:~# rmmod myext2
root@hao-zhang:~# |

```

五. 讨论、心得

1. 通过本次实验，我对 Linux 文件系统的文件和目录结构的理解，以及文件存储空间的管理、文件的物理结构和目录结构以及文件操作的实现有了更深的认识，特别是文件系统的模拟加深了我对文件系统内部功能和实现过程的理解；在设计文件系统实验中也遇到了一些小问题，但最后都解决了。
2. 使用系统调用函数实现简单文件备份的原理是：对于打开的源文件，逐字节读取并写入目标文件，循环直到文件结束。使用 C 语言库函数实现简单文件备份的原理是：对于打开的源文件，每次读固定数目的字节写入目标文件，循环直到文件结束。这两种方法基本是类似的，两者都是以只读方式打开源文件，以只写方式打开目标文件；主要区别在于一个是 Linux 操作系统提供的系统调用，一个 C 语言提供的库函数，且使用 C 语言库函数进行文件备份的速度较快。
3. 当系统没有显式提供打开文件的操作时，需要每次读文件或写文件时，都需要判断该文件是否已经打开，如果未打开，则打开该文件，否则就直接执行读写操作。相反，系统显式提供了打开文件的操作，就需要用户有打开文件的操作。
4. 示例代码中的 `gets()` 函数可以使用替代 `scanf()` 函数来替代。`scanf()` 函数的格式化字符串可以自己定义终止字符，格式为 `"%[^c]"`，其中 `c` 为任意字符，这样 `scanf()` 就可以读入带有空格的字符串了。但是要注意的一点是，`scanf` 并不会处理终止字符，所以需要使用 `getchar()` 函数来手动处理。
5. 设计文件系统实验中，编译内核模块时出错，经排查是使用的内核源码与当前系统的内核版本有细微差别，可以通过 `apt` 安装对应版本的内核源码解决。
6. 设计文件系统实验中编写了一些脚本文件，掌握了 shell 脚本的一些基本用法，包括循环、判断语句，`sed` 等。