浙江大学

本科实验报告

课程名称：操作系统

姓　　名：应承峻

学　　院：计算机科学与技术学院

系：计算机科学与技术系

专　　业：软件工程

学　　号：3170103456

指导教师：夏莹杰

2019 年 12 月 9 日

浙江大学实验报告

课程名称：操作系统 实验类型：综合

实验项目名称：添加一个加密文件系统

学生姓名：应承峻 专业：软件工程 学号：3170103456

电子邮件地址：3170103456@zju.edu.cn 手机：17326084929

实验地点：玉泉曹光彪西 503 实验日期：2019 年 12 月 9 日

一、实验目的和要求

文件系统是操作系统中最直观的部分，因为用户可以通过文件直接地和操作系统交互，操作系统也必须为用户提供数据计算、数据存储的功能。本实验通过添加一个文件系统，进一步理解 Linux 中的文件系统原理及其实现。

- 深入理解操作系统文件系统原理
- 学习理解 Linux 的 VFS 文件系统管理技术
- 学习理解 Linux 的 ext2 文件系统实现技术
- 设计和实现加密文件系统

二、实验内容

添加一个类似于 ext2，但对磁盘上的数据块进行加密的文件系统 myext2。实验主要内容：

- 添加一个类似 ext2 的文件系统 myext2
- 修改 myext2 的 magic number

- 添加文件系统创建工具
- 添加加密文件系统操作，包括 read_crypt，write_crypt，使其增加对加密数据的读写。

三、主要仪器设备

笔记本电脑 1 台，相关配置如下：

处理器　　英特尔 Core i7-8750H @ 2.20GHz 六核

内　存　　16 GB （三星 DDR4 2667MHz）

主硬盘　　PeM280240GP4C15B （240 GB/固态硬盘）

显　卡　　Nvidia GeForce GTX 1060（6 GB）

操作系统环境：Windows 10 64 位（DirectX 12）

Linux 版本：　ubuntu-19.04

四、操作方法和实验步骤

4.1 添加一个类似 ext2 的文件系统 myext2

要添加一个类似 ext2 的文件系统 myext2，首先是确定实现 ext2 文件系统的内核源码是由哪些文件组成。Linux 源代码结构很清楚地告诉我们：fs/ext2 目录下的所有文件是属于 ext2 文件系统的。再检查一下这些文件所包含的头文件，可以初步总结出来 Linux 源代码中属于 ext2 文件系统的有：

```
fs/ext2/acl.c
fs/ext2/acl.h
fs/ext2/balloc.c
fs/ext2/bitmap.c
fs/ext2/dir.c
fs/ext2/ext2.h
fs/ext2/file.c
.....
include/linux/ext2_fs.h
```

接下来开始添加 myext2 文件系统的源代码到 Linux 源代码。把 ext2 部分的源代码克隆到 myext2 去，即复制一份以上所列的 ext2 源代码文件给 myext2 用。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，头文件放到 include/linux 下。在 Linux 的 shell 下，执行如下操作：

```
#cd /usr/src/linux  /*内核源代码目录，假设内核源代码解压在主目录的 linux 子目录*/
#cd fs
#cp  -R ext2 myext2
#cd /usr/src/linux/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
```

这样就完成了克隆文件系统工作的第一步——源代码复制。对于克隆文件系统来说，这样当然还远远不够，因为文件里面的数据结构名、函数名、以及相关的一些宏等内容还没有根据 myext2 改掉，连编译都通不过。

下面开始克隆文件系统的第二步：修改上面添加的文件的内容。为了简单起见，做了一个最简单的替换：将原来“EXT2”替换成“MYEXT2”；将原来的“ext2”替换成“myext2”。

对于 fs/myext2 下面文件中字符串的替换，也可以使用下面的脚本：

```
#!/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

```
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
```

把这个脚本命名为 substitute.sh，放在 fs/myext2 下面，加上可执行权限，运行之后就可以把当前目录里所有文件里面的“ext2”和“EXT2”都替换成对应的“myext2”和“MYEXT2”。

特别提示：

- 不要拷贝 word 文档中的 substitute.sh 脚本，在 Linux 环境下重新输入一遍，substitute.sh 脚本程序只能运行一次。ubuntu 环境：sudo bash substitute.sh。
- 先删除 fs/myext2 目录下的 *.o 文件，再运行脚本程序。
- 在下面的替换或修改内核代码时可以使用 gedit 编辑器，要注意大小写。

用编辑器的替换功能，把 /lib/modules/$(uname -r)/build/include/linux/myext2_fs.h, 和 /lib/modules/$(uname -r)/build/include/asm-generic/bitops/ 下的 myext2-atomic.h 与 myext2-atomic-setbit.h 文件中的“ext2”、“EXT2”分别替换成“myext2”、“MYEXT2”

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

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

```
#include <asm-generic/bitops/le.h>
#include <asm-generic/bitops/ext2-atomic.h>
#include <asm-generic/bitops/myext2-atomic.h>a
```

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>

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

#include <asm-generic/bitops/ext2-atomic-setbit.h>
#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

```
#define EFS_SUPER_MAGIC         0x414A53
#define EXT2_SUPER_MAGIC        0xEF53
#define EXT3_SUPER_MAGIC        0xEF53
#define MYEXT2_SUPER_MAGIC      0xEF53
#define XENFS_SUPER_MAGIC       0xabba1974
```

源代码的修改工作到此结束。接下来就是第三步工作——把 myext2 编译成内核模块。

要编译内核模块，首先要生成一个 Makefile 文件。我们可以修改 myext2/Makefile 文件，

修改后的 Makefile 文件如下：

```
#
# 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
```

编译内核模块的命令是 make，在 myext2 目录下执行命令：

#make

```
root@ying:/usr/src/linux-5.3.5/fs/myext2# make
make -C /lib/modules/5.3.5/build M=/usr/src/linux-5.3.5/fs/myext2  modules
make[1]: 进入目录“/usr/src/linux-5.3.5”
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/balloc.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/dir.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/file.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/ialloc.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/inode.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/ioctl.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/namei.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/super.o
  CC [M]  /usr/src/linux-5.3.5/fs/myext2/symlink.o
  LD [M]  /usr/src/linux-5.3.5/fs/myext2/myext2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/src/linux-5.3.5/fs/myext2/myext2.mod.o
  LD [M]  /usr/src/linux-5.3.5/fs/myext2/myext2.ko
make[1]: 离开目录“/usr/src/linux-5.3.5”
root@ying:/usr/src/linux-5.3.5/fs/myext2#
```

编译好模块后，使用 insmod 命令加载模块：

#insmod myext2.ko

查看一下 myext2 文件系统是否加载成功：

#cat /proc/filesystems |grep myext2

```
root@ying:/usr/src/linux-5.3.5/fs/myext2# insmod myext2.ko
root@ying:/usr/src/linux-5.3.5/fs/myext2# cat /proc/filesystems |grep myext2
        myext2
root@ying:/usr/src/linux-5.3.5/fs/myext2#
```

确认 myext2 文件系统加载成功后，可以对添加的 myext2 文件系统进行测试了，输入命令 cd 先把当前目录设置成主目录。

对添加的 myext2 文件系统测试命令如下：

#dd if=/dev/zero of=myfs bs=1M count=1

#/sbin/mkfs.ext2 myfs

#mount -t myext2 -o loop ./myfs /mnt

#mount

……

…… on /mnt type myext2 (rw)

#umount /mnt

#mount -t ext2 -o loop ./myfs /mnt

#mount

……

…… on /mnt type ext2 (rw)

#umount /mnt

#rmmod myext2　/*卸载模块*/

```
ying@ying:~$ cd
ying@ying:~$ dd if=/dev/zero of=myfs bs=1M count=1
记录了1+0 的读入
记录了1+0 的写出
1048576 bytes (1.0 MB, 1.0 MiB) copied, 0.00104485 s, 1.0 GB/s
ying@ying:~$ /sbin/mkfs.ext2 myfs
mke2fs 1.44.6 (5-Mar-2019)
丢弃设备块：  完成
创建含有 1024 个块（每块 1k）和 128 个inode的文件系统

正在分配组表：  完成
正在写入inode表：  完成
写入超级块和文件系统账户统计信息：  已完成
```

```
ying@ying:~$ sudo mount -t myext2 -o loop ./myfs /mnt
[sudo] ying 的密码：
```

```
ying@ying:~$ mount | grep /mnt
/home/ying/myfs on /mnt type myext2 (rw,relatime,errors=continue)
ying@ying:~$
```

```
ying@ying:~$ sudo umount /mnt
ying@ying:~$ sudo mount -t ext2 -o loop ./myfs /mnt
ying@ying:~$ mount | grep /mnt
/home/ying/myfs on /mnt type ext2 (rw,relatime)
ying@ying:~$
```

4.2 修改 myext2 的 magic number

在上面做的基础上。找到 myext2 的 magic number，并将其改为 0x6666：4.6.0 内核版本，这个值在/usr/src/linux/include/uapi/linux/magic.h 文件中。

- #define MYEXT2_SUPER_MAGIC　0xEF53

+ #define MYEXT2_SUPER_MAGIC　0x6666

```
#define EFS_SUPER_MAGIC         0x414A53
#define EXT2_SUPER_MAGIC        0xEF53
#define EXT3_SUPER_MAGIC        0xEF53
#define MYEXT2_SUPER_MAGIC      0x6666
#define XENFS_SUPER_MAGIC       0xabba1974
#define EXT4_SUPER_MAGIC        0xEF53
#define BTRFS_SUPER_MAGIC       0x9123683E
#define NILFS_SUPER_MAGIC       0x3434
```

改动完成之后，再用 make 重新编译内核模块，使用命令 insmod 安装编译好的 myext2.ko 内核模块。

在我们测试这个部分之前，我们需要写个小程序 changeMN.c，来修改我们创建的 myfs 文件系统的 magic number。因为它必须和内核中记录 myext2 文件系统的 magic number 匹配，myfs 文件系统才能被正确地 mount。

changeMN.c 程序可以在课程网站中下载。这个程序经过编译后产生的可执行程序名字为 changeMN。

```
ying@ying:~$ gcc -o changeMN changeMN.c
changeMN.c:2:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
```

下面我们开始测试：

#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

…… on /mnt type myext2 (rw)

#sudo umount /mnt

sudo mount -t ext2 -o loop ./fs.new /mnt

mount: wrong fs type, bad option, bad superblock on /dev/loop0, …

rmmod myext2

```
ying@ying:~$ ./changeMN myfs
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
ying@ying:~$
```

```
ying@ying:~$ sudo mount -t myext2 -o loop ./fs.new /mnt
ying@ying:~$ mount | grep /mnt
/home/ying/fs.new on /mnt type myext2 (rw,relatime,errors=continue)
ying@ying:~$
```

```
ying@ying:~$ sudo umount /mnt
ying@ying:~$ sudo mount -t ext2 -o loop ./fs.new /mnt
mount: /mnt: wrong fs type, bad option, bad superblock on /dev/loop15, missing c
odepage or helper program, or other error.
ying@ying:~$
```

4.3 修改文件系统操作

myext2 只是一个实验性质的文件系统，我们希望它只要能支持简单的文件操作即可。因此在完成了 myext2 的总体框架以后，我们来修改掉 myext2 支持的一些操作，来加深对操作系统对文件系统的操作的理解。下面以裁减 myext2 的 mknod 操作为例，了解这个过程的实现流程。

Linux 将所有的对块设备、字符设备和命名管道的操作，都看成对文件的操作。mknod 操作是用来产生那些块设备、字符设备和命名管道所对应的节点文件。在 ext2 文件系统中它的实现函数如下：

```
fs/ext2/namei.c, line 141
static int ext2_mknod (struct inode * dir, struct dentry *dentry, int mode,
dev_t rdev)
{
        struct inode * inode;
        int err;

        if (!new_valid_dev(rdev))
                return -EINVAL;

        inode = ext2_new_inode (dir, mode);
        err = PTR_ERR(inode);
        if (!IS_ERR(inode)) {
                init_special_inode(inode, inode->i_mode, rdev);
#ifdef CONFIG_EXT2_FS_XATTR
                inode->i_op = &ext2_special_inode_operations;
#endif
```

```
                mark_inode_dirty(inode);
                err = ext2_add_nondir(dentry, inode);
        }
        return err;
}
```

它定义在结构 ext2_dir_inode_operations 中：

```
fs/ext2/namei.c, line 428
struct inode_operations ext2_dir_inode_operations = {
        .create         = ext2_create,
        .lookup         = ext2_lookup,
        .link           = ext2_link,
        .unlink         = ext2_unlink,
        .symlink        = ext2_symlink,
        .mkdir          = ext2_mkdir,
        .rmdir          = ext2_rmdir,
        .mknod          = ext2_mknod,
        .rename         = ext2_rename,
#ifdef CONFIG_EXT2_FS_XATTR
        .setxattr       = generic_setxattr,
        .getxattr       = generic_getxattr,
        .listxattr      = ext2_listxattr,
        .removexattr    = generic_removexattr,
#endif
        .setattr        = ext2_setattr,
        .permission     = ext2_permission,
};
```

当然，从 ext2 克隆过去的 myext2 的 myext2_mknod，以及 myext2_dir_inode_operations 和上面的程序是一样的。对于 mknod 函数，我们在 myext2 中作如下修改：

fs/myext2/namei.c

```
static int myext2_mknod (struct inode * dir, struct dentry *dentry, int mode, int rdev)
{
   printk(KERN_ERR “haha, mknod is not supported by myext2! you’ve been cheated!\n”);
   return -EPERM;
   /*
   …..
    把其它代码注释
   */
}
```

添加的程序中：

第一行 打印信息，说明 mknod 操作不被支持。

第二行 将错误号为 EPERM 的结果返回给 shell，即告诉 shell，在 myext2 文件系统中，mknod 不被支持。

```
static int myext2_mknod (struct inode * dir, struct dentry *dentry, umode_t mode, dev_t rdev)
{
        printk(KERN_ERR "haha, mknod is not supported by myext2! you've been cheated!\n");
        return -EPERM;
}
```

修改完毕，再用 make 重新编译内核模块，使用命令 insmod 安装编译好的 myext2.ko 内核模块。我们在 shell 下执行如下测试程序：

```
#mount -t myext2 -o loop ./fs.new /mnt
#cd /mnt
#mknod myfifo p
    mknod: `myfifo': Operation not permitted
#
```

第一行命令：将 fs.new mount 到/mnt 目录下。

第二行命令：进入/mnt 目录，也就是进入 fs.new 这个 myext2 文件系统。

第三行命令：执行创建一个名为 myfifo 的命名管道的命令。

第四、五行是执行结果：第四行是我们添加的 myext2_mknod 函数的 printk 的结果；第五行

是返回错误号 EPERM 结果给 shell，shell 捕捉到这个错误后打出的出错信息。需要注意的是，如果你是在图形界面下使用虚拟控制台，printk 打印出来的信息不一定能在你的终端上显示出来，但是可以通过命令 dmesg|tail 来观察。

可见，我们的裁减工作取得了预期的效果。

```
ying@ying:/mnt$ sudo mknod myfifo p
mknod: myfifo: 不允许的操作
ying@ying:/mnt$ dmesg | grep "haha"
[ 2228.157875] haha, mknod is not supported by myext2! you've been cheated!
ying@ying:/mnt$
```

4.4. 添加文件系统创建工具

文件系统的创建对于一个文件系统来说是首要的。因为，如果不存在一个文件系统，所有对它的操作都是空操作，也是无用的操作。

其实，前面的第一小节《添加一个类似 ext2 的文件系统 myext2》和第二小节《修改 myext2 的 magic number》在测试实验结果的时候，已经陆陆续续地讲到了如何创建 myext2 文件系统。下面工作的主要目的就是将这些内容总结一下，制作出一个更快捷方便的 myext2 文件系统的创建工具：mkfs.myext2（名称上与 mkfs.ext2 保持一致）。

首先需要确定的是该程序的输入和输出。为了灵活和方便起见，我们的输入为一个文件，这个文件的大小，就是 myext2 文件系统的大小。输出就是带了 myext2 文件系统的文件。

我们在主目录下编辑如下的程序：

~/mkfs.myext2

```
#!/bin/bash

/sbin/losetup -d /dev/loop2
/sbin/losetup /dev/loop2 $1
/sbin/mkfs.ext2 /dev/loop2
dd if=/dev/loop2 of=./tmpfs bs=1k count=2
./changeMN $1 ./tmpfs
dd if=./fs.new of=/dev/loop2
/sbin/losetup -d /dev/loop2
  rm -f ./tmpfs
```

第一行 表明是 shell 程序。

第三行 如果有程序用了/dev/loop2 了，就将它释放。

第四行 用 losetup 将第一个参数代表的文件装到/dev/loop2 上

第五行 用 mkfs.ext2 格式化/dev/loop2。也就是用 ext2 文件系统格式格式化我们的文件系统。

第六行 将文件系统的头 2K 字节的内容取出来，复制到 tmpfs 文件里面。

第七行 调用程序 changeMN 读取 tmpfs，复制到 fs.new，并且将 fs.new 的 magic number 改成 0x6666

第八行 再将 2K 字节的内容写回去。

第九行 把我们的文件系统从 loop2 中卸下来。

第十行 将临时文件删除。

我们发现 mkfs.myext2 脚本中的 changeMN 程序功能，与 2.2 节的 changeMN 功能不一样，请修改 changeMN.c 程序，以适合本节 mkfs.myext2 和下面测试的需要。

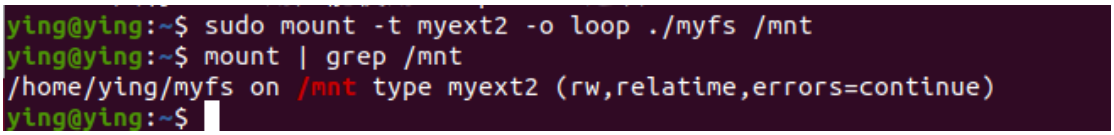
```
#include <stdio.h>
main()
{
        int ret;
        FILE *fp_read;
        FILE *fp_write;
        unsigned char buf[2048];

        fp_read=fopen("./tmpfs","rb");

        if(fp_read == NULL)
        {
                printf("open myfs failed!\n");
                return 1;
        }
```

编辑完了之后，做如下测试：

```
# dd if=/dev/zero of=myfs bs=1M count=1
# ./mkfs.myext2 myfs   (或 sudo bash mkfs.myext2 myfs )
# mount -t myext2 -o loop ./myfs /mnt
# mount
    /dev/loop on /mnt myext2 (rw)
```



4.5 修改加密文件系统的 read 和 write 操作

在内核模块 myext2.ko 中修改 file.c 的代码，添加两个函数 new_sync_read_crypt 和 new_sync_write_crypt，将这两个函数指针赋给 myext2_file_operations 结构中的 read 和 write 操作。在 new_sync_write_crypt 中增加对用户传入数据 buf 的加密，在 new_sync_read_crypt 中增加解密。可以使用 DES 等加密和解密算法。

对 new_sync_read_cryp 函数，可以做如下修改：

```
ssize_t new_sync_write_crypt(struct file *filp, const char __user *buf, size_t len, loff_t *ppos)
{
        char* mybuf = buf;
        //在此处添加对长度为 len 的 buf 数据进行加密（简单移位密码，将每个字符值+25）

        printk("haha encrypt %ld\n", len);
```

return new_sync_write(filp, mybuf, len, ppos);//调用默认的写函数，把加密数据写入

}

对 new_sync_read_cryp 函数，可以做如下修改：

ssize_t new_sync_read_crypt(struct file *filp, char __user *buf, size_t len, loff_t *ppos)

{

int i;

//先调用默认的读函数读取文件数据

ssize_t ret = new_sync_read(filp, buf, len, ppos);

//此处添加对文件的解密（简单移位解密，将每个字符值-25）

printk("haha encrypt %ld\n", len);

return ret;

}

```
ssize_t new_sync_write_crypt(struct file *filp, const char __user *buf, size_t len, loff_t *ppos)
{
        char* mybuf = (char*)kmalloc(sizeof(char)*len,GFP_KERNEL);
        int i;
        copy_from_user(mybuf, buf, len);
        for (i=0;i<len;i++) {
                mybuf[i] = (mybuf[i] + 25) % 128;
        }
        copy_to_user(buf, mybuf, len);
        printk("haha encrypt %ld\n", len);
        return new_sync_write(filp, buf, len, ppos);
}

ssize_t new_sync_read_crypt(struct file *filp, char __user *buf, size_t len, loff_t *ppos)
{
        int i;
        char* mybuf = (char*)kmalloc(sizeof(char)*len,GFP_KERNEL);
        ssize_t ret = new_sync_read(filp, buf, len, ppos);
        copy_from_user(mybuf, buf, len);
        for (i=0;i<len;i++) mybuf[i] = (mybuf[i] - 25 + 128) % 128;
        copy_to_user(buf, mybuf, len);
        printk("haha encrypt %ld\n", len);
        return ret;
}
```

上述修改完成后，再用 make 重新编译 myext2 模块，使用命令 insmod 安装编译好的 myext2.ko 内核模块。重新加载 myext2 内核模块，创建一个 myext2 文件系统，并尝试往文件系统中写入一个字符串文件。

mount -t myext2 -o loop ./fs.new /mnt/

cd /mnt/

新建文件 test.txt 并写入字符串“1234567”，再查看 test.txt 文件内容：cat test.txt 。

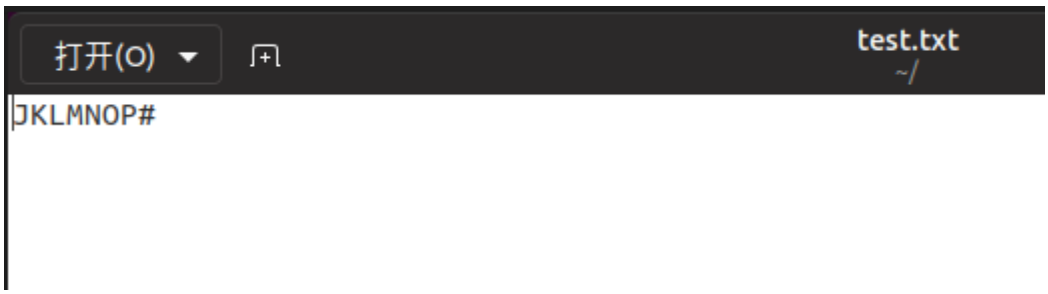
```
ying@ying:/mnt$ cat test.txt
1234567
ying@ying:/mnt$
```

把 test.txt 文件复制到主目录下：cp test.txt ~ 。

在主目录下打开 test.txt 文件，查看 test.txt 文件内容的结果？

```
ying@ying:/mnt$ cd -
/home/ying
ying@ying:~$ cat test.txt
1234567
ying@ying:~$
```

使用文件管理器的复制，再查看结果？



我们把之前的 magic number 改回 0xEF53。重新编译 myext2 模块，安装 myext2.ko 后，执行下面命令：53

dd if=/dev/zero of=myfs bs=1M count=1

/sbin/mkfs.ext2 myfs

mount -t myext2 -o loop ./myfs /mnt

cd /mnt

echo “1234567” > test.txt

cat test.txt

cd

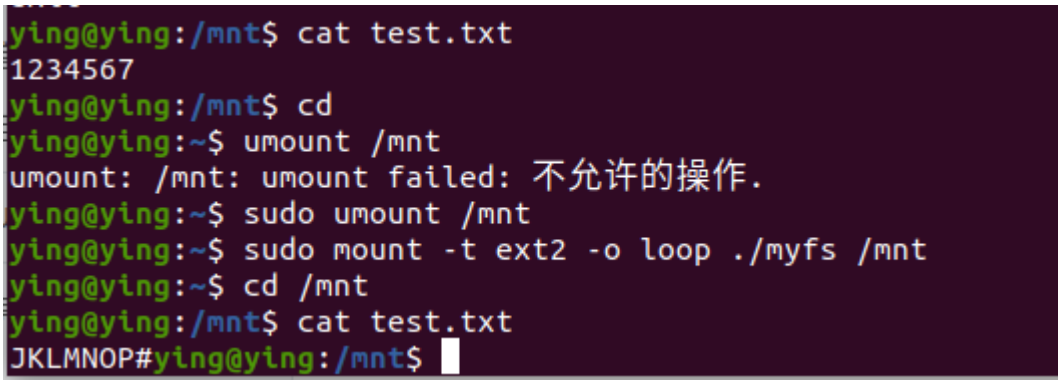
umount /mnt

mount -t ext2 -o loop ./myfs /mnt

cd /mnt

cat test.txt

查看实验结果，此时即使使用 ext2 文件系统的 magic number，在 myext2 文件系统中创建的文件都是加密文件。



五、讨论和心得

通过本次实验，我学习了如何修改文件系统的内核，并熟悉了文件系统实现的一些细节，在实验过程中主要遇到了如下问题：

1. 一开始在 make 的时候出现了下图问题，查看报错信息应该是宏定义造成的问题，因此排查发现漏了以下三个文件的修改，使用编辑器批量替换即可。

/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

```
ying@ying:/usr/src/linux/fs/myext2$ make
make -C /lib/modules/4.6.0/build M=/usr/src/linux-4.6/fs/myext2  modules
make[1]: Entering directory '/usr/src/linux-4.6'
  CC [M]  /usr/src/linux-4.6/fs/myext2/balloc.o
In file included from include/linux/linkage.h:4:0,
                 from include/linux/fs.h:4,
                 from /usr/src/linux-4.6/fs/myext2/myext2.h:13,
                 from /usr/src/linux-4.6/fs/myext2/balloc.c:14:
/usr/src/linux-4.6/fs/myext2/myext2.h: In function ‘verify_offsets’:
/usr/src/linux-4.6/fs/myext2/myext2.h:637:4: error: ‘MYEXT2_SB_MAGIC_OFFSET’ und
eclared (first use in this function)
  A(MYEXT2_SB_MAGIC_OFFSET, s_magic);
    ^
include/linux/compiler.h:486:19: note: in definition of macro ‘__compiletime_ass
ert’
   bool __cond = !(condition);    \
                  ^
include/linux/compiler.h:506:2: note: in expansion of macro ‘_compiletime_assert
’
  _compiletime_assert(condition, msg, __compiletime_assert_, __LINE__)
  ^
```

2. 在测试的过程中，遇到 module 卸载不掉或者是 umount 时为 busy 的情况，解决方法是先关闭/mnt 那个进程，然后 umount，最后 rmmod。

```
root@ying:/usr/src/linux-4.6/fs/myext2# rmmod myext2
rmmod: ERROR: Module myext2 is in use
root@ying:/usr/src/linux-4.6/fs/myext2# umount /mnt
```

3. 在 Linux 4.1 版本后 new_sync_write 和 new_sync_read 这两个函数变成了静态函数，无法直接在 file.c 文件中调用，因此在参考了附录[1]的教程后，重新编译内核，即可使用。

六、附录

[1] https://blog.csdn.net/qq_33040649/article/details/85640676