

浙江大学实验报告

课程名称： 操作系统 实验类型： 综合型

实验项目名称： 添加一个加密系统

学生姓名： 刘轩铭 学号： 3180106071

电子邮件地址： 519102931@qq.com

实验日期： 2020 年 12 月 21 日

一、 实验环境

- 阿里云服务器
- Ubuntu16.04
- Linux-4.8.0

二、 实验内容和结果及分析

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。

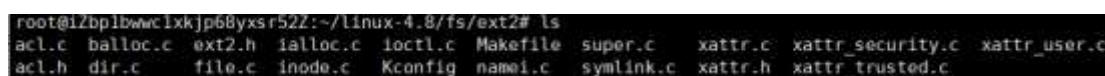
三、 实验步骤

3.1 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 myext2 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
```

查看内容如图所示：



```
root@i2bp1bwwc1xkjp68yxr522:~/linux-4.8/fs/ext2# ls  
acl.c  balloc.c  ext2.h  ialloc.c  ioctl.c  Makefile  super.c  xattr.c  xattr_security.c  xattr_user.c  
acl.h  dir.c    file.c  inode.c  Kconfig  namei.c  symlink.c  xattr.h  xattr_trusted.c
```

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 ~/linux-4.8.0  
  
#cd fs  
  
#cp -R ext2 myext2  
  
#cd ~/linux-3.18.24/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
```

```
root@iZbp1bw1xkjp68yxs52Z:~# cd linux-4.8/
root@iZbp1bw1xkjp68yxs52Z:~/linux-4.8# cd fs
root@iZbp1bw1xkjp68yxs52Z:~/linux-4.8/fs# cp -R ext2 myext2
root@iZbp1bw1xkjp68yxs52Z:~/linux-4.8/fs# cd myext2/
root@iZbp1bw1xkjp68yxs52Z:~/linux-4.8/fs/myext2# mv ex2.h myext2.h
mv: cannot stat 'ex2.h': No such file or directory
root@iZbp1bw1xkjp68yxs52Z:~/linux-4.8/fs/myext2# mv ext2.h myext2.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
```

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。

运行该脚本：


```

* For the benefit of those who are trying to port Linux to another
* architecture, here are some C-language equivalents.  You should
* recode these in the native assembly language, if at all possible.
*
* C language equivalents written by Theodore Ts'o, 9/26/92
*/

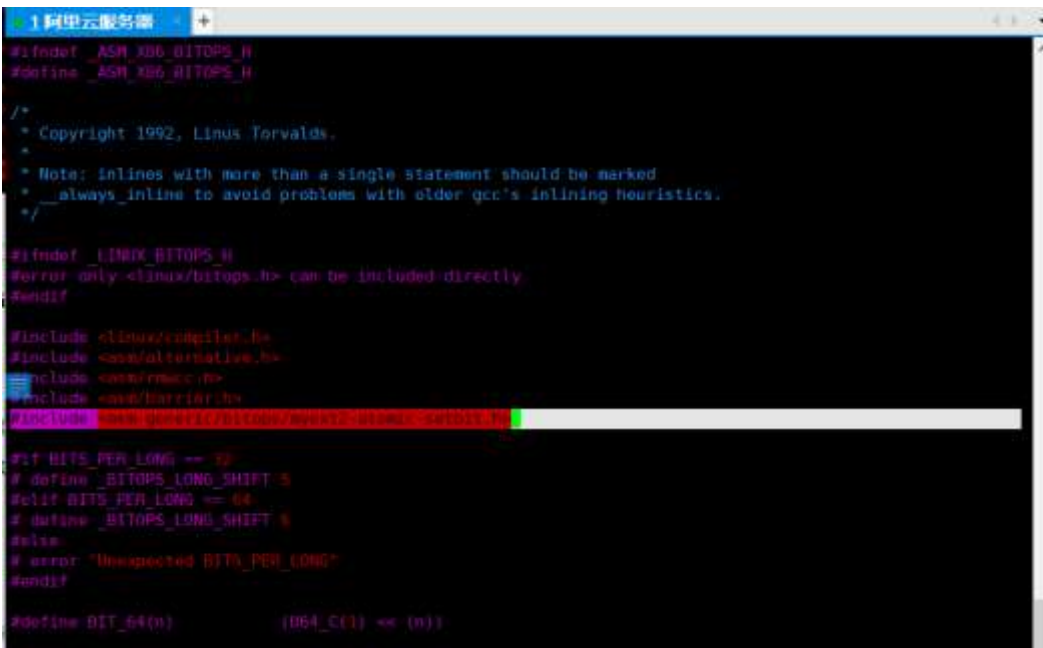
#include <linux/irqflags.h>
#include <linux/compiler.h>
#include <asm/barrier.h>

#include <asm-generic/bitops/__ffs.h>
#include <asm-generic/bitops/ffz.h>
#include <asm-generic/bitops/fls.h>
#include <asm-generic/bitops/__fls.h>
#include <asm-generic/bitops/fls64.h>
#include <asm-generic/bitops/find.h>
#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>
```



```

1 阿里云服务器
#define ASM_X86_BITOPS_H
#define _ASM_X86_BITOPS_H

/*
 * Copyright 1992, Linus Torvalds.
 *
 * Note: Inlines with more than a single statement should be marked
 * __always_inline to avoid problems with older gcc's inlining heuristics.
 */

#ifndef LINUX_BITOPS_H
#error only <linux/bitops.h> can be included directly
#endif

#include <linux/compiler.h>
#include <asm/alternative.h>
#include <asm/rmcc.h>
#include <asm/barrier.h>
#include <asm-generic/bitops/myext2-atomic-setbit.h>

#if BITS_PER_LONG == 32
# define _BITOPS_LONG_SHIFT 5
#elif BITS_PER_LONG == 64
# define _BITOPS_LONG_SHIFT 6
#else
# error "Unexpected BITS_PER_LONG"
#endif

#define BIT_64(n) ((u64_C1) << (n))

```

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

```
#define MYEXT2_SUPER_MAGIC 0xEF53
```

```
1 阿里云服务器 +
#define __LINUX_MAGIC_H__
#define __LINUX_MAGIC_H__

// 32
#define MYEXT2_SUPER_MAGIC 0x5346414F
#define ADFS_SUPER_MAGIC 0xadf5
#define AFFS_SUPER_MAGIC 0xaadf5f
#define AFS_SUPER_MAGIC 0x5346414F
#define AUTofs_SUPER_MAGIC 0xad1b7
#define CODA_SUPER_MAGIC 0x73757245
#define CRAMFS_MAGIC 0x28cd3d44 /* some random number */
#define CRAMFS_MAGIC_WEND 0x453dc020 /* magic number with the wrong endianness */
#define DEBUGFS_MAGIC 0x64626260
#define SECURITYFS_MAGIC 0x73636673
#define SELINUX_MAGIC 0xf97c1f8c
#define SMACK_MAGIC 0x43415053 /* "SMAC" */
#define RAMFS_MAGIC 0x85b458f5 /* some random number */
#define TMPFS_MAGIC 0x01021994
#define HURDIBFS_MAGIC 0x958458f6 /* some random number */
#define SQUASHFS_MAGIC 0x73717368
#define ECRYPTFS_SUPER_MAGIC 0x115f
#define EFS_SUPER_MAGIC 0x414a53
#define EXT2_SUPER_MAGIC 0xef53
#define EXT3_SUPER_MAGIC 0xef53
#define XENFS_SUPER_MAGIC 0xa0b61974
#define EXT4_SUPER_MAGIC 0xef53
#define BTRFS_SUPER_MAGIC 0x9123683e
#define NILFS_SUPER_MAGIC 0x3434
#define F2FS_SUPER_MAGIC 0xaf2f52010
#define HPPFS_SUPER_MAGIC 0x1995a049
```

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@iZbp1bwmc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# make
make -C /lib/modules/4.8.0/build M=/root/linux-4.8/fs/myext2 modules
make[1]: Entering directory '/root/linux-4.8'
CC [M] /root/linux-4.8/fs/myext2/balloc.o
CC [M] /root/linux-4.8/fs/myext2/dir.o
CC [M] /root/linux-4.8/fs/myext2/file.o
CC [M] /root/linux-4.8/fs/myext2/ialloc.o
CC [M] /root/linux-4.8/fs/myext2/inode.o
CC [M] /root/linux-4.8/fs/myext2/ioctl.o
CC [M] /root/linux-4.8/fs/myext2/namei.o
CC [M] /root/linux-4.8/fs/myext2/super.o
CC [M] /root/linux-4.8/fs/myext2/symlink.o
LD [M] /root/linux-4.8/fs/myext2/mymyext2.o
Building modules, stage 2.
MODPOST 1 modules
CC /root/linux-4.8/fs/myext2/mymyext2.mod.o
LD [M] /root/linux-4.8/fs/myext2/mymyext2.ko
make[1]: Leaving directory '/root/linux-4.8'

```

编译好模块后，使用 insmod 命令加载模块：

```
#insmod myext2.ko
```

```

root@iZbp1bwmc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# insmod myext2.ko
root@iZbp1bwmc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# insmod myext2.ko

```

查看一下 myext2 文件系统是否加载成功：

```
#cat /proc/filesystems |grep myext2
```

```

root@iZbp1bwmc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# cat /proc/filesystems | grep myext2
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*/
```

```
root@iZbp1bwclxkjp68yxsr52Z:~# 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, 0.00140678 s, 745 MB/s
```

```
root@iZbp1bwclxkjp68yxsr52Z:~# /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@iZbp1bwclxkjp68yxsr52Z:~# mount -t myext2 -o loop ./myfs /mnt
root@iZbp1bwclxkjp68yxsr52Z:~# 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=863248k,nr_inodes=215812,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=204800k,mode=755)
/dev/vda1 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/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
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/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=30,pgrp=1,timeout=0,minproto=5,maxproto=
,direct,pipe_ino=10588)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
```

```
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,relatime)
tmpfs on /run/user/0 type tmpfs (rw,nosuid,nodev,relatime,size=204800k,mode=700)
/root/myfs on /mnt type myext2 (rw,relatime,errors=continue)
```

```

root@iZbp1bwclxkjp68yxs52Z:~# mount -t ext2 -o loop ./myfs /mnt
root@iZbp1bwclxkjp68yxs52Z:~# 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=863248k,nr_inodes=215812,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=264888k,mode=755)
/dev/vda1 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/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
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/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=30,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=19588)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,relatime)
tmpfs on /run/user/0 type tmpfs (rw,nosuid,nodev,relatime,size=204800k,mode=700)
/root/myfs on /mnt type ext2 (rw,relatime,block validity,barrier,user_xattr,acl)

```

```

root@iZbp1bwclxkjp68yxs52Z:~# umount /mnt
root@iZbp1bwclxkjp68yxs52Z:~# rmdir myext2
root@iZbp1bwclxkjp68yxs52Z:~#

```

3.2 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：

4.8.0 内核版本，这个值在 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
// lxm
#define MYEXT2_SUPER_MAGIC     0x6666
#define EXT3_SUPER_MAGIC       0xEF53
#define XENFS_SUPER_MAGIC      0xabba1974

```

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。

```

root@iZbp1bwxc1xkjp68yxs52Z:~/linux-4.8/fs/myext2# make
make -C /lib/modules/4.8.0/build M=/root/linux-4.8/fs/myext2 modules
make[1]: Entering directory '/root/linux-4.8'
CC [M] /root/linux-4.8/fs/myext2/balloc.o
CC [M] /root/linux-4.8/fs/myext2/dir.o
CC [M] /root/linux-4.8/fs/myext2/file.o
CC [M] /root/linux-4.8/fs/myext2/ialloc.o
CC [M] /root/linux-4.8/fs/myext2/inode.o
CC [M] /root/linux-4.8/fs/myext2/ioctl.o
CC [M] /root/linux-4.8/fs/myext2/namei.o
CC [M] /root/linux-4.8/fs/myext2/super.o
CC [M] /root/linux-4.8/fs/myext2/symlink.o
LD [M] /root/linux-4.8/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
LD [M] /root/linux-4.8/fs/myext2/myext2.ko
make[1]: Leaving directory '/root/linux-4.8'
root@iZbp1bwxc1xkjp68yxs52Z:~/linux-4.8/fs/myext2# insmod myext2.ko

```

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。

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

```

这里与书上不一样的

```

..... 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

```

```
root@iZbp1bwwclxkjp68yxsr52Z:~# 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, 0.00147225 s, 712 MB/s
```

```
root@iZbp1bwwclxkjp68yxsr52Z:~# 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, 0.0015595 s, 672 MB/s
root@iZbp1bwwclxkjp68yxsr52Z:~# /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@iZbp1bwwclxkjp68yxsr52Z:~# gcc changeMN.c -o changMN
changeMN.c:9:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^
root@iZbp1bwwclxkjp68yxsr52Z:~# ./changMN myfs
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
```

```
root@iZbp1bwwclxkjp68yxsr52Z:~# mount -t myext2 -o loop ./fs.new /mnt
root@iZbp1bwwclxkjp68yxsr52Z:~# 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=863248k,nr_inodes=215812,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=204800k,mode=755)
/dev/vdal 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=/
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/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
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/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=30,pgrp=1,timeout=0,minproto=5,max
proto=5,direct,pipe_ino=10588)
mqueue on /dev/mqueue type mqueue (rw,relatime)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,relatime)
tmpfs on /run/user/0 type tmpfs (rw,nosuid,nodev,relatime,size=204800k,mode=700)
/root/fs.new on /mnt type myext2 (rw,relatime,errors=continue)
```

```
root@iZbplbwwc1xkjp68yxsr52Z:~# sudo umount /mnt
root@iZbplbwwc1xkjp68yxsr52Z:~# sudo 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.
```

这里由于我们修改了 magic number，所以挂载 ext2 失败。

```
root@iZbplbwwc1xkjp68yxsr52Z:~# rmmod myext2
```

2.3 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 144

```
144 static int ext2_mknod (struct inode *dir, struct dentry *dentry, int mode, dev_t
rdev)
145 {
146     struct inode * inode;
147     int err;
148
149     if (!new_valid_dev(rdev))
150         return -EINVAL;
151
152     inode = ext2_new_inode (dir, mode);
153     err = PTR_ERR(inode);
```



```

154         if (!IS_ERR(inode)) {
155             init_special_inode(inode, inode->i_mode, rdev);
156 #ifdef CONFIG_EXT2_FS_XATTR
157             inode->i_op = &ext2_special_inode_operations;
158 #endif
159             mark_inode_dirty(inode);
160             err = ext2_add_nondir(dentry, inode);
161         }
162         return err;
163 }

```

它定义在结构 `ext2_dir_inode_operations` 中：

`fs/ext2/namei.c`, line 400

```

392 struct inode_operations ext2_dir_inode_operations = {
393     .create      = ext2_create,
394     .lookup      = ext2_lookup,
395     .link        = ext2_link,
396     .unlink      = ext2_unlink,
397     .symlink     = ext2_symlink,
398     .mkdir       = ext2_mkdir,
399     .rmdir       = ext2_rmdir,
400     .mknod       = ext2_mknod,
401     .rename      = ext2_rename,
402 #ifdef CONFIG_EXT2_FS_XATTR
403     .setxattr    = generic_setxattr,
404     .getxattr    = generic_getxattr,
405     .listxattr   = ext2_listxattr,
406     .removexattr = generic_removexattr,
407 #endif

```

```

408         .setattr          = ext2_setattr,
409         .permission        = ext2_permission,
410     };

```

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
    */
}

```



```

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;
    /*
    struct inode * inode;
    int err;

    err = dqquot_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_nodir(dentry, inode);
    }
    return err;
    */
}

```

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。

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。

```
root@iZbp1bwclxkjp68yxsr52Z:~/linux-4.8/fs/myext2# make
make -C /lib/modules/4.8.0/build M=/root/linux-4.8/fs/myext2 modules
make[1]: Entering directory '/root/linux-4.8'
  CC [M]  /root/linux-4.8/fs/myext2/namei.o
  LD [M]  /root/linux-4.8/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /root/linux-4.8/fs/myext2/myext2.mod.o
  LD [M]  /root/linux-4.8/fs/myext2/myext2.ko
make[1]: Leaving directory '/root/linux-4.8'
root@iZbp1bwclxkjp68yxsr52Z:~/linux-4.8/fs/myext2# insmod myext2.ko

root@iZbp1bwclxkjp68yxsr52Z:~# cd /mnt
root@iZbp1bwclxkjp68yxsr52Z:/mnt# mknod myfifo p
mknod: myfifo: Operation not permitted

[1543069.259524] EXT4-fs (loop0): VFS: Can't find ext4 filesystem
[1547393.733205] haha, mknod is not supported by myext2! you've been cheated!
```

可见，我们的裁减工作取得了预期的效果。

3.4. 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《添加一个和类似 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
```

这里与教材上不一样的，以本实验指导为准。

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。

```

/*
 * Disclaimer: Possible loss of data!! This code was written by the students
 * who took the Operating System module at Zhejiang University. Use it at
 * your own will and risk! If you do not know what the code does, please try
 * not to use it.
 */

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

    fp_read=fopen("./tmpfs","rb"); // change from "./myfs" to "./tmpfs"

    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 )

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

# mount

/dev/loop on /mnt myext2 (rw)

```

```

root@iZbp1bwclxkjp68yxsr52Z:~# 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, 0.00150151 s, 698 MB/s

```

```

root@iZbp1bwclxkjp68yxsr52Z:~# 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, 0.00150151 s, 698 MB/s
root@iZbp1bwclxkjp68yxsr52Z:~# sudo bash 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, 0.0110027 s, 186 kB/s
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
4+0 records in
4+0 records out
2048 bytes (2.0 kB, 2.0 KiB) copied, 0.0142468 s, 144 kB/s

```

```

root@iZbp1bwclxkjp68yxsr52Z:~# sudo mount -t myext2 -o loop ./myfs /mnt
root@iZbp1bwclxkjp68yxsr52Z:~# 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=863248k,nr_inodes=215812,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=204800k,mode=755)
/dev/vda1 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/cpuset type cgroup (rw,nosuid,nodev,noexec,relatime,cpuset)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/blkio type cgroup (rw,nosuid,nodev,noexec,relatime,blkio)
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/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/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
systemd-1 on /proc/sys/fs/binfmt_misc type autofs (rw,relatime,fd=23,pgrp=1,timeout=0,minproto=5,maxproto=5,direct,pipe_ino=10578)
hugetlbfs on /dev/hugepages type hugetlbfs (rw,relatime)
mqueue on /dev/mqueue type mqueue (rw,relatime)
debugfs on /sys/kernel/debug type debugfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,relatime)
tmpfs on /run/user/0 type tmpfs (rw,nosuid,nodev,relatime,size=204800k,mode=700)
/root/fs.new on /mnt type myext2 (rw,relatime,errors=continue)
/root/myfs on /mnt type myext2 (rw,relatime,errors=continue)

```

3.5 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_rmmt` 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。（//以 3.18 内核为例）

```
/*
 * We have mostly NULL's here: the current defaults are ok for
 * the myext2 filesystem.
 */
const struct file_operations myext2_file_operations = {
    .llseek      = generic_file_llseek,
    // lxm
    .read_iter   = generic_file_read_iter_crypt,
    .write_iter  = generic_file_write_iter_crypt,
    // .read_iter = generic_file_read_iter,
    // .write_iter = generic_file_write_iter,
    .unlocked_ioctl = myext2_ioctl,
#ifdef CONFIG_COMPAT
    .compat_ioctl = myext2_compat_ioctl,
#endif
    .mmap        = myext2_file_mmap,
    .open        = dquot_file_open,
    .release     = myext2_release_file,
    .fsync       = myext2_fsync,
    .splice_read = generic_file_splice_read,
    .splice_write = iter_file_splice_write,
};

const struct inode_operations myext2_file_inode_operations = {
#ifdef CONFIG_MYEXT2_FS_XATTR
    .setxattr    = generic_setxattr,
    .getxattr    = generic_getxattr,
    .listxattr   = myext2_listxattr,
    .removexattr = generic_removexattr,
#endif
    .setattr     = myext2_setattr,
    .get_acl     = myext2_get_acl,
    .set_acl     = myext2_set_acl,
    .fiemap      = myext2_fiemap,
};
```

对 `new_sync_write_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_crypt 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;
}
```

```
#include <linux/time.h>
#include <linux/pagemap.h>
#include <linux/dax.h>
#include <linux/quotaops.h>
#include <linux/uio.h>
#include "myext2.h"
#include "xattr.h"
#include "acl.h"
```

```
// lxm
ssize_t generic_file_write_iter_crypt(struct kiocb *iocb, struct iov_iter *iter)
{
    int i;
    int len = iter->iov->iov_len;
    char* d = kmalloc(len, GFP_USER);

    copy_from_user(d, iter->iov->iov_base, len);
    for(i = 0; i < len; i++)
        d[i] = (d[i] + 25) % 128;
    copy_to_user(iter->iov->iov_base, d, len);

    printk("haha encrypt %ld\n", len);
    return generic_file_write_iter(iocb, iter);
}

ssize_t generic_file_read_iter_crypt(struct kiocb *iocb, struct iov_iter *iter)
{
    int i;
    int len = iter->iov->iov_len;
    char* d = kmalloc(len, GFP_USER);
    ssize_t value = generic_file_read_iter(iocb, iter);

    copy_from_user(d, iter->iov->iov_base, len);
    for(i = 0; i < len; i++)
        d[i] = (d[i] - 25 + 128) % 128;
    copy_to_user(iter->iov->iov_base, d, len);

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

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 ./myfs /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`。

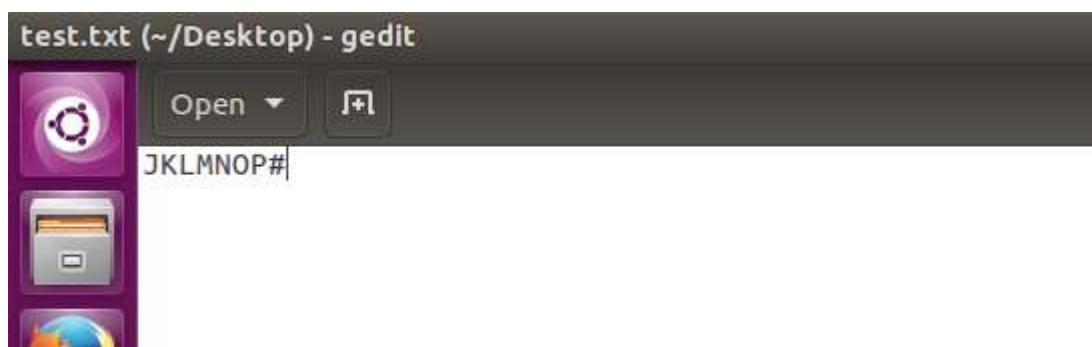
```
root@iZbp1bwwc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# make  
make -C /lib/modules/4.8.0/build M=/root/linux-4.8/fs/myext2 modules  
make[1]: Entering directory '/root/linux-4.8'  
CC [M] /root/linux-4.8/fs/myext2/file.o  
/root/linux-4.8/fs/myext2/file.c: In function 'generic_file_write_iter_crypt':  
/root/linux-4.8/fs/myext2/file.c:172:12: warning: format '%ld' expects argument of type 'long int', but  
printk("haha encrypt %ld\n", len);  
^  
/root/linux-4.8/fs/myext2/file.c: In function 'generic_file_read_iter_crypt':  
/root/linux-4.8/fs/myext2/file.c:188:12: warning: format '%ld' expects argument of type 'long int', but  
printk("haha encrypt %ld\n", len);  
^  
LD [M] /root/linux-4.8/fs/myext2/myext2.o  
Building modules, stage 2.  
MODPOST 1 modules  
CC /root/linux-4.8/fs/myext2/myext2.mod.o  
LD [M] /root/linux-4.8/fs/myext2/myext2.ko  
make[1]: Leaving directory '/root/linux-4.8'  
root@iZbp1bwwc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# insmod myext2.ko
```

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

```
root@iZbp1bwwc1xkjp68yxsr52Z:~# cat test.txt  
1234567
```

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：

```
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。

```
#define EFS_SUPER_MAGIC      0x414A53
#define EXT2_SUPER_MAGIC     0xEF53
// lxm
#define MYEXT2_SUPER_MAGIC    0xEF53
#define EXT2_SUPER_MAGIC     0xEF53
```



```

root@iZbp1bwwc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# insmod myext2.ko
root@iZbp1bwwc1xkjp68yxsr52Z:~/linux-4.8/fs/myext2# cd ~
root@iZbp1bwwc1xkjp68yxsr52Z:~# 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, 0.00131448 s, 798 MB/s
root@iZbp1bwwc1xkjp68yxsr52Z:~# /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@iZbp1bwwc1xkjp68yxsr52Z:~# mount -t myext2 -o loop ./myfs /mnt
root@iZbp1bwwc1xkjp68yxsr52Z:~# cd /mnt/
root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# echo "1234567" > test.txt
root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# cat test.txt
1234567
root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# cd
root@iZbp1bwwc1xkjp68yxsr52Z:~# umount /mnt
root@iZbp1bwwc1xkjp68yxsr52Z:~# mount -t ext
ext2 ext3 ext4
root@iZbp1bwwc1xkjp68yxsr52Z:~# mount -t ext2 -o loop ./myfs /mnt
root@iZbp1bwwc1xkjp68yxsr52Z:~# cd /m
-bash: cd: /m: No such file or directory
root@iZbp1bwwc1xkjp68yxsr52Z:~# cd /mnt
root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# cat test.txt
JKLMNOP#root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# cat test.txt
JKLMNOP#root@iZbp1bwwc1xkjp68yxsr52Z:/mnt# █

```

可以看到，此时显示的内容为加密过的内容。

至此，文件系统部分的实验已经全部完成了。通过本实验，你对 Linux 整个文件系统的运作流程，如何添加一个文件系统，以及如何修改 Linux 对文件系统的操作，有了比较深的了解。在本实验的基础上，你完全可以发挥自己的创造性，构造出自己的文件系统，然后将它添加到 Linux 中。

从结果分析，myext2 不但支持原来 ext2 文件系统的部分操作，还添加了用加密数据进行读写的操作，对 1234567 进行加密，在用 ext2 文件格式读取后，结果为 JKLMNOP，实现了对数据的加密读写。

3.6 源代码

修改 file.c:

```

1. #include <linux/uio.h>
2.
3. // lxm

```



```

4. ssize_t generic_file_write_iter_crypt(struct kiocb *iocb, struct iov_iter *i
   ter)
5. {
6.     int i;
7.     int len = iter->iov->iov_len;
8.     char* d = kmalloc(len, GFP_USER);
9.
10.    copy_from_user(d, iter->iov->iov_base, len);
11.    for(i = 0; i < len; i++)
12.        d[i] = (d[i] + 25) % 128;
13.    copy_to_user(iter->iov->iov_base, d, len);
14.
15.    printk("haha encrypt %ld\n", len);
16.    return generic_file_write_iter(iocb, iter);
17. }
18.
19. ssize_t generic_file_read_iter_crypt(struct kiocb *iocb, struct iov_iter *it
   er)
20. {
21.     int i;
22.     int len = iter->iov->iov_len;
23.     char* d = kmalloc(len, GFP_USER);
24.     ssize_t value = generic_file_read_iter(iocb, iter);
25.
26.    copy_from_user(d, iter->iov->iov_base, len);
27.    for(i = 0; i < len; i++)
28.        d[i] = (d[i] - 25 + 128) % 128;
29.    copy_to_user(iter->iov->iov_base, d, len);
30.
31.    printk("haha encrypt %ld\n", len);
32.    return value;
33. }
34.
35.
36. /*
37.  * We have mostly NULL's here: the current defaults are ok for
38.  * the myext2 filesystem.
39.  */
40. const struct file_operations myext2_file_operations = {
41.     .llseek      = generic_file_llseek,
42.     // lxm
43.     .read_iter   = generic_file_read_iter_crypt,
44.     .write_iter  = generic_file_write_iter_crypt,
45.     // .read_iter = generic_file_read_iter,

```

```

46. // .write_iter = generic_file_write_iter,
47. .unlocked_ioctl = myext2_ioctl,
48. #ifdef CONFIG_COMPAT
49. .compat_ioctl = myext2_compat_ioctl,
50. #endif
51. .mmap = myext2_file_mmap,
52. .open = dquot_file_open,
53. .release = myext2_release_file,
54. .fsync = myext2_fsync,
55. .splice_read = generic_file_splice_read,
56. .splice_write = iter_file_splice_write,
57. };

```

substitute.sh:

```

1. #!/bin/bash
2.
3. SCRIPT=substitute.sh
4.
5. for f in *
6. do
7.     if [ $f = $SCRIPT ]
8.     then
9.         echo "skip $f"
10.        continue
11.    fi
12.
13.    echo -n "substitute ext2 to myext2 in $f..."
14.    cat $f | sed 's/ext2/myext2/g' > ${f}_tmp
15.    mv ${f}_tmp $f
16.    echo "done"
17.
18.    echo -n "substitute EXT2 to MYEXT2 in $f..."
19.    cat $f | sed 's/EXT2/MYEXT2/g' > ${f}_tmp
20.    mv ${f}_tmp $f
21.    echo "done"
22.
23. done

```

修改后的 changeMN.c:

```

1. /*

```

```
2.  * Disclaimer: Possible loss of data!! This code was written by the students
3.  * who took the Operating System module at Zhejiang University. Use it at
4.  * your own will and risk! If you do not know what the code does, please try
5.  * not to use it.
6.  */
7.
8. #include <stdio.h>
9. main()
10. {
11.     int ret;
12.     FILE *fp_read;
13.     FILE *fp_write;
14.     unsigned char buf[2048];
15.
16.     fp_read=fopen("./tmpfs","rb"); // change from "./myfs" to "./tmpfs"
17.
18.     if(fp_read == NULL)
19.     {
20.         printf("open myfs failed!\n");
21.         return 1;
22.     }
23.
24.     fp_write=fopen("./fs.new","wb");
25.
26.     if(fp_write==NULL)
27.     {
28.         printf("open fs.new failed!\n");
29.         return 2;
30.     }
31.
32.     ret=fread(buf,sizeof(unsigned char),2048,fp_read);
33.
34.     printf("previous magic number is 0x%x%x\n",buf[0x438],buf[0x439]);
35.
36.     buf[0x438]=0x66;
37.     buf[0x439]=0x66;
38.
39.     fwrite(buf,sizeof(unsigned char),2048,fp_write);
40.
41.     printf("current magic number is 0x%x%x\n",buf[0x438],buf[0x439]);
42.
43.     while(ret == 2048)
```

```

44.     {
45.         ret=fread(buf,sizeof(unsigned char),2048,fp_read);
46.         fwrite(buf,sizeof(unsigned char),ret,fp_write);
47.     }
48.
49.     if(ret < 2048 && feof(fp_read))
50.     {
51.         printf("change magic number ok!\n");
52.     }
53.
54.     fclose(fp_read);
55.     fclose(fp_write);
56.
57.     return 0;
58. }

```

四、 讨论与心得

这个实验最难的部分在于理解各个部分的内容和命令，而不是简单的跟着实验指导进行。

在第五步的时候，我碰到了许多问题。首先，由于内核版本的问题，原文最后调用的系统函数是 `new_sync_read` 和 `new_sync_write`，但 Linux 4.1 版本后这两个函数变成了静态函数（见 `fs/read_write.c`），无法直接在我们写的 `file.c` 文件中调用。相关内容可以见 Linux 4.1 修改日志：

（<https://github.com/torvalds/linux/commit/5d5d568975307877e9195f5305f4240e506a2807>）。为了解决这个问题，我在网上找了许多的信息。最终采用了改写系统提供的 `generic_file_write/read_iter` 函数的方法。这一个问题解决后，我又碰到了另外的问题。在挂载文件系统的时候，出现了如下的报错：

```

root@iZbp1bwclxkjp68yxsr52Z:~# mount -t myext2 -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.

```

我思考了很久，最终将挂载点改成 `myfs`，然后解决了问题。为了解决这个问题，我甚至换到了虚拟机上进行实验，发现还会有更多奇怪的问题。为了解决问题，我也大量查阅了资料。

此外，我也明白了 `myext2` 是 `ext2` 的定制版本，支持原来 `ext2` 文件系统的部

分操作，还添加了用加密数据进行读写的操作。

在做这个实验的时候，我还对 Linux 的虚拟文件系统查阅了相关资料。理解了 Linux 内核支持装载不同的文件系统类型，不同的文件系统有各自管理文件的方式，所以引入了 vfs，来进行简化。