

浙江大学实验报告

课程名称：____操作系统分析及实验____实验类型：____设计性____
实验项目名称：____实验3 添加一个加密文件系统____
学生姓名：____蒋仕彪____专业：____求是科学班 1701____学号：3170102587____
电子邮件地址：____969519450@qq.com____手机：____18888921530____
实验日期：2019 年 12 月 19 日

一、实验目的

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、实验器材

安装了 Vmware 的 Windows10 计算机。

虚拟机中的 Linux 为 Ubuntu-64，自带的内核版本为 4.15.0。在实验二时安装了 4.6.0 的内核，本次实验将在 4.6.0 的内核上展开。

四、操作方法和实验步骤

4.1 创建新的文件系统 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 myext 告诉我们：**fs/ext2** 目录下的所有文件是属于 ext2 文件系统的。
- 复制一份 ext2 源代码文件作为 myext2。具体地，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。

- 复制和改名的 shell 命令如下：

```
jsb@ubuntu: /lib/modules/4.6.0
jsb@ubuntu:~$ uname -r
4.6.0
jsb@ubuntu:~$ cd /usr/src/linux/fs
jsb@ubuntu:/usr/src/linux/fs$ sudo cp -r ext2 ./myext2
[sudo] password for jsb:
jsb@ubuntu:/usr/src/linux/fs$ cd myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ mv ext2.h myext2.h
mv: cannot move 'ext2.h' to 'myext2.h': Permission denied
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo mv ext2.h myext2.h
```

- 在调用库里注册一个新的头文件 myext2.h:

```
jsb@ubuntu: /lib/modules/4.6.0/build/include/asm-generic/bitops
jsb@ubuntu:~$ cd /lib/modules/$(uname -r)/build/include/linux
jsb@ubuntu:/lib/modules/4.6.0/build/include/linux$ sudo cp ext2_fs.h myext2_fs.h
jsb@ubuntu:/lib/modules/4.6.0/build/include/linux$ cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops
jsb@ubuntu:/lib/modules/4.6.0/build/include/asm-generic/bitops$ sudo cp ext2-atomic.h myext2-atomic.h
jsb@ubuntu:/lib/modules/4.6.0/build/include/asm-generic/bitops$ sudo cp ext2-atomic-setbit.h myext2-atomic-setbit.h
jsb@ubuntu:/lib/modules/4.6.0/build/include/asm-generic/bitops$
```

4.2 替换新文件系统的内容

- 注意到我们只是复制了一份 ext2 并改名为 myext2, 代码里面调用的名字还是 ext2。所以要写一个 shell 命令去修改所有文件里面的内容。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”。

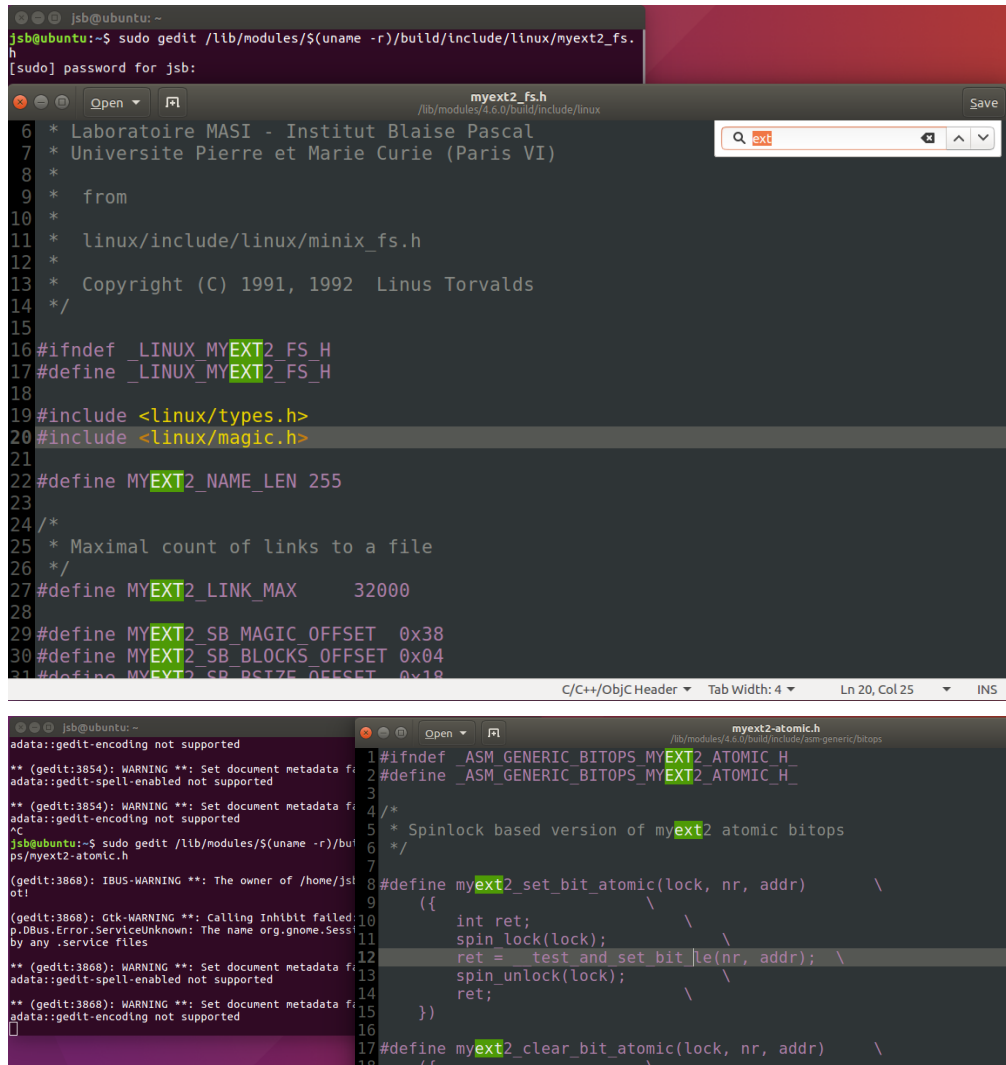
```
substitute.sh
#!/bin/bash
SCRIPT=substitute.sh
for f in *
do
if [ $f = $SCRIPT ]
then
echo "skip $f"
continue
fi
echo -n "substitute ext2 to myext2 in $f..."
cat $f | sed 's/ext2/myext2/g' > ${f}_tmp
mv ${f}_tmp $f
echo "done"
echo -n "substitute EXT2 to MYEXT2 in $f..."
cat $f | sed 's/EXT2/MYEXT2/g' > ${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”。

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo bash substitute.sh
[sudo] password for jsb:
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 ballo.c...done
substitute EXT2 to MYEXT2 in ballo.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 lallo.c...done
substitute EXT2 to MYEXT2 in lallo.c...done
substitute ext2 to myext2 in lnode.c...done
substitute EXT2 to MYEXT2 in lnode.c...done
substitute ext2 to myext2 in lnode.c...done
substitute EXT2 to MYEXT2 in lnode.c...done
substitute ext2 to myext2 in locl.c...done
substitute EXT2 to MYEXT2 in locl.c...done
substitute ext2 to myext2 in kconfi...done
substitute EXT2 to MYEXT2 in kconfi...done
substitute ext2 to myext2 in kconfi...done
substitute EXT2 to MYEXT2 in kconfi...done
substitute ext2 to myext2 in Makefile...done
substitute EXT2 to MYEXT2 in Makefile...done
substitute ext2 to myext2 in myext2...done
substitute EXT2 to MYEXT2 in myext2...done
```

4.3 为新文件系统添加头文件

- 注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`。

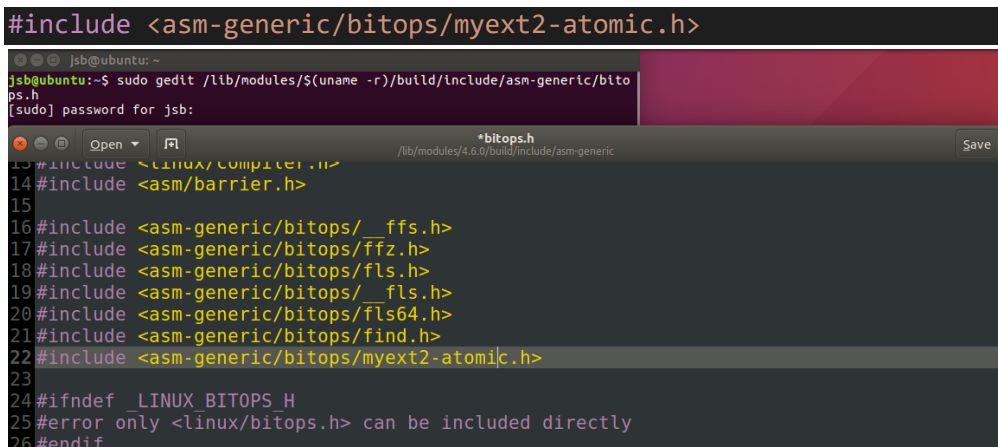


```
jsb@ubuntu:~$ sudo gedit /lib/modules/$(uname -r)/build/include/linux/myext2_fs.h
[sudo] password for jsb:

myext2_fs.h
6 * Laboratoire MASI - Institut Blaise Pascal
7 * Universite Pierre et Marie Curie (Paris VI)
8 *
9 * from
10 *
11 * linux/include/linux/minix_fs.h
12 *
13 * Copyright (C) 1991, 1992 Linus Torvalds
14 */
15
16 #ifndef _LINUX_MYEXT2_FS_H
17 #define _LINUX_MYEXT2_FS_H
18
19 #include <linux/types.h>
20 #include <linux/magic.h>
21
22 #define MYEXT2_NAME_LEN 255
23
24 /*
25  * Maximal count of links to a file
26  */
27 #define MYEXT2_LINK_MAX 32000
28
29 #define MYEXT2_SB_MAGIC_OFFSET 0x38
30 #define MYEXT2_SB_BLOCKS_OFFSET 0x04
31 #define MYEXT2_SB_BTREE_OFFSET 0x18

myext2-atomic.h
1 #ifndef _ASM_GENERIC_BITOPS_MYEXT2_ATOMIC_H
2 #define _ASM_GENERIC_BITOPS_MYEXT2_ATOMIC_H
3
4 /*
5  * Spinlock based version of myext2 atomic bitops
6  */
7
8 #define myext2_set_bit_atomic(lock, nr, addr) \
9 ({ \
10     int ret; \
11     spin_lock(lock); \
12     ret = test_and_set_bit_le(nr, addr); \
13     spin_unlock(lock); \
14     ret; \
15 })
16
17 #define myext2_clear_bit_atomic(lock, nr, addr) \
18 ({ \
```

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



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

jsb@ubuntu:~$ sudo gedit /lib/modules/$(uname -r)/build/include/asm-generic/bitops.h
[sudo] password for jsb:

*bitops.h
13 #include <linux/compiler.h>
14 #include <asm/barrier.h>
15
16 #include <asm-generic/bitops/ffs.h>
17 #include <asm-generic/bitops/ffz.h>
18 #include <asm-generic/bitops/fls.h>
19 #include <asm-generic/bitops/fls64.h>
20 #include <asm-generic/bitops/find.h>
21 #include <asm-generic/bitops/myext2-atomic.h>
22
23
24 #ifndef _LINUX_BITOPS_H
25 #error only <linux/bitops.h> can be included directly
26 #endif
```

- 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
```

```
jsb@ubuntu:~$ sudo gedit /lib/modules/$(uname -r)/build/include/uapi/linux/magic.h
```

```
magic.h
19 #define ECRIFS_SUPER_MAGIC 0xA1D1
20 #define EFS_SUPER_MAGIC 0x414A53
21 #define EXT2_SUPER_MAGIC 0xEF53
22 #define MYEXT2_SUPER_MAGIC 0xEF53
23 #define EXT3_SUPER_MAGIC 0xEF53
24 #define XENFS_SUPER_MAGIC 0xabba1974
25 #define EXT4_SUPER_MAGIC 0xEF53
26 #define BTRFS_SUPER_MAGIC 0x9123683E
27 #define NILFS_SUPER_MAGIC 0x3434
```

4.4 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:

```
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
```

- 输入 make 命令后，程序报错:

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo make
[sudo] password for jsb:
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
/usr/src/linux-4.6/fs/myext2/balloc.c: In function 'myext2_free_blocks':
/usr/src/linux-4.6/fs/myext2/balloc.c:539:8: error: implicit declaration of function 'myext2_clear_bit_atomic' [-Werror=implicit-function-declaration]
    if (!myext2_clear_bit_atomic(sb_bgl_lock(sbi, block_group),
    ^
/usr/src/linux-4.6/fs/myext2/balloc.c: In function 'myext2_try_to_allocate':
/usr/src/linux-4.6/fs/myext2/balloc.c:716:6: error: implicit declaration of function 'myext2_set_bit_atomic' [-Werror=implicit-function-declaration]
    if (myext2_set_bit_atomic(sb_bgl_lock(MYEXT2_SB(sb), group), grp_goal,
    ^
cc1: some warnings being treated as errors
scripts/Makefile.build:291: recipe for target '/usr/src/linux-4.6/fs/myext2/balloc.o' failed
make[2]: *** [/usr/src/linux-4.6/fs/myext2/balloc.o] Error 1
Makefile:1428: recipe for target '_module_/usr/src/linux-4.6/fs/myext2' failed
make[1]: *** [_module_/usr/src/linux-4.6/fs/myext2] Error 2
make[1]: Leaving directory '/usr/src/linux-4.6'
Makefile:11: recipe for target 'default' failed
make: *** [default] Error 2
```

- 根据报错信息，应该有两处地方关于 myext2 的函数未定义。
- 结合报错信息和之前的步骤，发现之前有一处文件忘记添加.h，添加后编译成功。

```

jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo 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
CC [M] /usr/src/linux-4.6/fs/myext2/dir.o
CC [M] /usr/src/linux-4.6/fs/myext2/file.o
CC [M] /usr/src/linux-4.6/fs/myext2/ialloc.o
CC [M] /usr/src/linux-4.6/fs/myext2/inode.o
CC [M] /usr/src/linux-4.6/fs/myext2/ioctl.o
CC [M] /usr/src/linux-4.6/fs/myext2/namei.o
CC [M] /usr/src/linux-4.6/fs/myext2/super.o
CC [M] /usr/src/linux-4.6/fs/myext2/symlink.o
LD [M] /usr/src/linux-4.6/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
CC /usr/src/linux-4.6/fs/myext2/myext2.mod.o
LD [M] /usr/src/linux-4.6/fs/myext2/myext2.ko
make[1]: Leaving directory '/usr/src/linux-4.6'

```

- 使用 insmod 命令加载模块并查看一下 myext2 文件系统是否加载成功。

```

jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo insmod myext2.ko
jsb@ubuntu:/usr/src/linux/fs/myext2$ cat /proc/filesystems |grep myext2
myext2

```

可以发现，myext2 模块已经正确加载了。

- 对 myext2 模块进行测试（后来的步骤里有相同内容的测试，这一步我就略过了）。

4.5 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 include/uapi/linux/magic.h 文件中）。

```

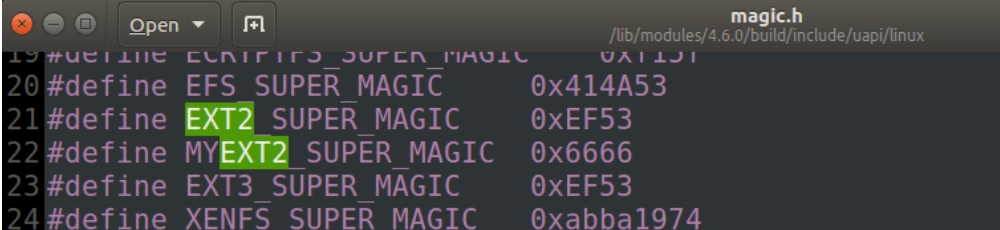
- #define MYEXT2_SUPER_MAGIC    0xEF53
+ #define MYEXT2_SUPER_MAGIC    0x6666

```

```

jsb@ubuntu: ~
jsb@ubuntu:~$ sudo gedit /lib/modules/$(uname -r)/build/include/uapi/linux/magic.h
[sudo] password for jsb:

```



```

19 #define ECKPTFS_SUPER_MAGIC    0x1131
20 #define EFS_SUPER_MAGIC        0x414A53
21 #define EXT2_SUPER_MAGIC        0xEF53
22 #define MYEXT2_SUPER_MAGIC      0x6666
23 #define EXT3_SUPER_MAGIC        0xEF53
24 #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。然后把老师提供的 changeMN.c 的小程序移动进去编译。

```

jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo gcc /home/jsb/Desktop/changeMN.c -o changeMN
[sudo] password for jsb:
/home/jsb/Desktop/changeMN.c:2:1: warning: return type defaults to 'int' [-Wimplicit-int]
main()
^

```

编译成功，并把可执行文件 changeMN 放在主目录下。

- 开始测试我们的新文件系统:

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

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo gcc /home/jsb/Desktop/changeMN.c -o changeMN
[sudo] password for jsb:
/home/jsb/Desktop/changeMN.c:2:1: warning: return type defaults to 'int' [-Wimplicit-int]
    main()
    ^
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo 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, 0.00872409 s, 120 MB/s
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo /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

jsb@ubuntu: /usr/src/linux/fs/myext2$ ./changeMN
open fs.new failed!
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo ./changeMN
previous magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
jsb@ubuntu: /usr/src/linux/fs/myext2$
```

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo mount -t myext2 -o loop ./fs.new /mnt
jsb@ubuntu: /usr/src/linux/fs/myext2$ 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=993124k,nr_inodes=248281,node=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,node=620)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=202920k,node=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,node=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/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/cpu,cpuacct type cgroup (rw,nosuid,nodev,noexec,relatime,cpu,cpuacct)
cgroup on /sys/fs/cgroup/pids type cgroup (rw,nosuid,nodev,noexec,relatime,pids)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory)
cgroup on /sys/fs/cgroup/freezer type cgroup (rw,nosuid,nodev,noexec,relatime,freezer)
cgroup on /sys/fs/cgroup/perf_event type cgroup (rw,nosuid,nodev,noexec,relatime,perf_event)
cgroup on /sys/fs/cgroup/hugetlb type cgroup (rw,nosuid,nodev,noexec,relatime,hugetlb)
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)
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=12071)
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)
configfs on /sys/kernel/config type configfs (rw,relatime)
fusectl on /sys/fs/fuse/connections type fusectl (rw,relatime)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock (rw,relatime,user_id=0,group_id=0,default_permissions,allow_other)
tmpfs on /run/user/1000 type tmpfs (rw,nosuid,nodev,relatime,size=202920k,node=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)
/usr/src/linux-4.6/fs/myext2/fs.new on /mnt type myext2 (rw,relatime,errors=continue)
jsb@ubuntu: /usr/src/linux/fs/myext2$
```

```
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo umount /mnt
jsb@ubuntu: /usr/src/linux/fs/myext2$ 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.
jsb@ubuntu: /usr/src/linux/fs/myext2$ rmmod myext2
rmmod: ERROR: ../libkmod/libkmod-module.c:793 knod_module_remove_module() could not remove 'myext2': Operation not permitted
rmmod: ERROR: could not remove module myext2: Operation not permitted
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo rmmod myext2
jsb@ubuntu: /usr/src/linux/fs/myext2$
```

从以上截图可以看到，整一套测试很顺利。

4.6 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;
}
```

- 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：

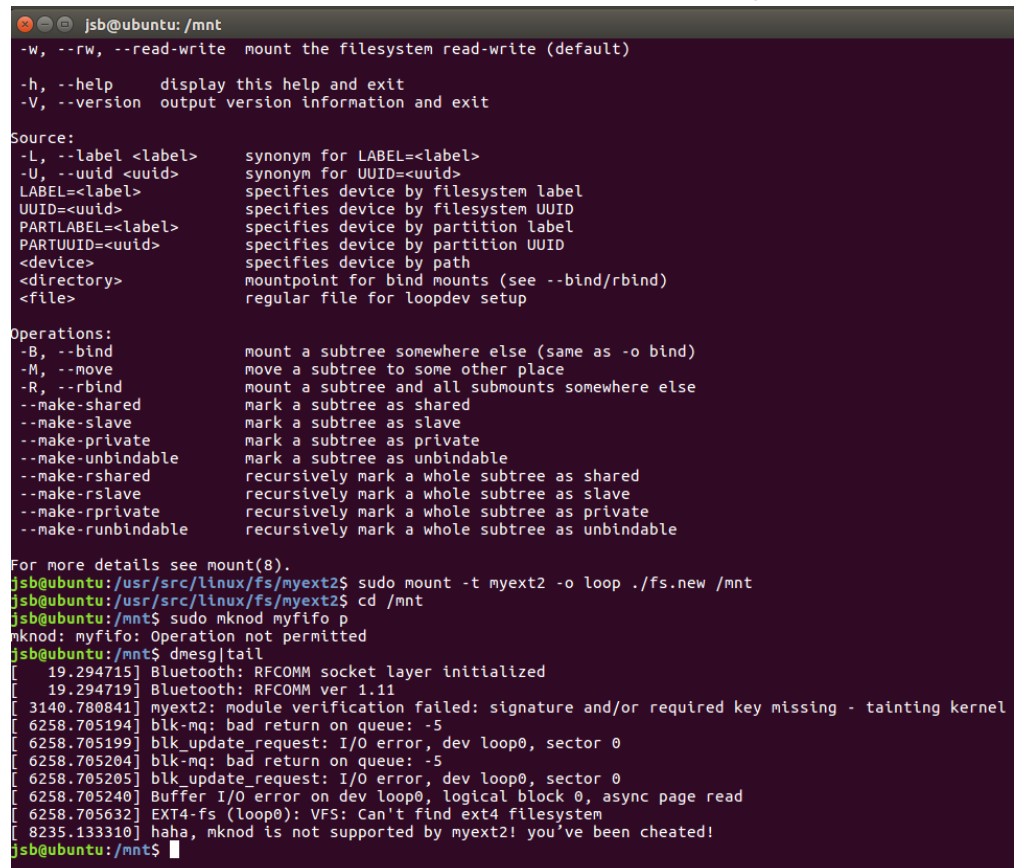


```
namei.c
140
141 static int myext2_mknod (struct inode * dir, struct dentry *dentry, umode_t mode, dev_t rdev)
142 {
143     printk(KERN_ERR "haha, mknod is not supported by myext2! you've been cheated!\n");
144     return -EPERM;
145
146     /*struct inode * inode;
147     int err;
148
149     err = dquot_initialize(dir);
150     if (err)
151         return err;
152
153     inode = myext2_new_inode (dir, mode, &dentry->d_name);
154     err = PTR_ERR(inode);
155     if (!IS_ERR(inode)) {
156         init_special_inode(inode, inode->i_mode, rdev);
157 #ifdef CONFIG_MYEXT2_FS_XATTR
158         inode->i_op = &myext2_special_inode_operations;
159 #endif
160         mark_inode_dirty(inode);
161         err = myext2_add_nondir(dentry, inode);
162     }
163     return err;*/
164 }
```

- 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。
- 修改完毕后，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 shell 下执行如下测试程序：

```
mount -t myext2 -o loop ./fs.new /mnt
cd /mnt
mknod myfifo p
```

- 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。



```
jsb@ubuntu: /mnt
-w, --rw, --read-write  mount the filesystem read-write (default)

-h, --help      display this help and exit
-V, --version   output version information and exit

Source:
-L, --label <label>      synonym for LABEL=<label>
-U, --uuid <uuid>        synonym for UUID=<uuid>
LABEL=<label>             specifies device by filesystem label
UUID=<uuid>               specifies device by filesystem UUID
PARTLABEL=<label>        specifies device by partition label
PARTUUID=<uuid>          specifies device by partition UUID
<device>                 specifies device by path
<directory>              mountpoint for bind mounts (see --bind/rbind)
<file>                   regular file for loopdev setup

Operations:
-B, --bind              mount a subtree somewhere else (same as -o bind)
-M, --move              move a subtree to some other place
-R, --rbind              mount a subtree and all submounts somewhere else
--make-shared            mark a subtree as shared
--make-slave             mark a subtree as slave
--make-private           mark a subtree as private
--make-unbindable        mark a subtree as unbindable
--make-rshared            recursively mark a whole subtree as shared
--make-rslave            recursively mark a whole subtree as slave
--make-rprivate          recursively mark a whole subtree as private
--make-runbindable       recursively mark a whole subtree as unbindable

For more details see mount(8).
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo mount -t myext2 -o loop ./fs.new /mnt
jsb@ubuntu: /usr/src/linux/fs/myext2$ cd /mnt
jsb@ubuntu: /mnt$ sudo mknod myfifo p
mknod: myfifo: Operation not permitted
jsb@ubuntu: /mnt$ dmesg|tail
[ 19.294715] Bluetooth: RFCOMM socket layer initialized
[ 19.294719] Bluetooth: RFCOMM ver 1.11
[ 3140.780841] myext2: module verification failed: signature and/or required key missing - tainting kernel
[ 6258.705194] blk-mq: bad return on queue: -5
[ 6258.705199] blk_update_request: I/O error, dev loop0, sector 0
[ 6258.705204] blk-mq: bad return on queue: -5
[ 6258.705205] blk_update_request: I/O error, dev loop0, sector 0
[ 6258.705240] Buffer I/O error on dev loop0, logical block 0, async page read
[ 6258.705632] EXT4-fs (loop0): VFS: Can't find ext4 filesystem
[ 8235.133310] haha, mknod is not supported by myext2! you've been cheated!
jsb@ubuntu: /mnt$
```

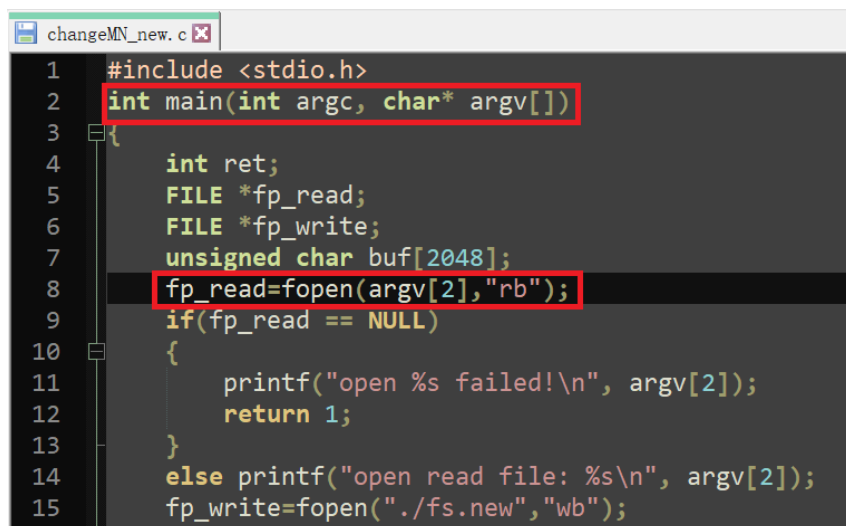
- 刚开始测试时，当我输入 `sudo mount -t myext2 -o loop ./fs.new /mnt` 的指令后，该指令没有正常执行，还跳出一个形如 `mount -help` 的界面（见上图上面部分）。重复了好几次后依然是这个错误。
- 为什么会出现 help 界面呢？应该是输入格式有问题！研究了一下我发现，这个命令我是直接从实验指导的 words 里复制的，有格式问题。手输了一遍就解决了。

4.7 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:

```
#!/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_new $1 ./tmpfs
dd if=./fs.new of=/dev/loop2
/sbin/losetup -d /dev/loop2
rm -f ./tmpfs
```

- mkfs.myext2 脚本中的 changeMN 程序功能, 与 2.2 节的 changeMN 功能不一样, 需要我们自行修改 changeMN.c 程序, 以适合本节 mkfs.myext2 和下面测试的需要。
- 原理很简单, 给该 C 程序加上参数 (且观察发现, 第二个参数是程序的输入)。



```
changeMN_new.c
1  #include <stdio.h>
2  int main(int argc, char* argv[])
3  {
4      int ret;
5      FILE *fp_read;
6      FILE *fp_write;
7      unsigned char buf[2048];
8      fp_read=fopen(argv[2],"rb");
9      if(fp_read == NULL)
10     {
11         printf("open %s failed!\n", argv[2]);
12         return 1;
13     }
14     else printf("open read file: %s\n", argv[2]);
15     fp_write=fopen("./fs.new","wb");
```

- 编辑完了之后, 做如下测试:

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

- 测试结果如下图:

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo gcc /home/jsb/Desktop/changeMN_new.c -o changeMN_new
[sudo] password for jsb:
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo cp /home/jsb/Desktop/mkfs.myext2 ./
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo 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, 0.00735956 s, 142 MB/s
jsb@ubuntu: /usr/src/linux/fs/myext2$ 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, 0.00788045 s, 260 kB/s
open read file: ./tmpfs
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, 0.00445361 s, 460 kB/s
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo mount -t myext2 -o loop ./myfs /mnt
jsb@ubuntu: /usr/src/linux/fs/myext2$ 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=993124k,nr_inodes=248281,mode=755)
devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620)
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,size=202920k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,data=ordered)
```

4.8 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。
- 在 file.c 里写入加密算法和解密算法，代码如下：

```
41 */
42
43 ssize_t new_sync_write_crypt(struct file *filp, const char __user *buf, size_t len, loff_t *ppos)
44 {
45     char* mybuf = (char*)kmalloc(sizeof(char)*len,GFP_KERNEL);
46     int i;
47     copy_from_user(mybuf,buf,len);
48
49     for(i=0;i<len;i++){
50         mybuf[i] = ((mybuf[i] + 25) % 128) ^ 127;
51     }
52     copy_to_user((void *)buf,mybuf,len);
53     printk("haha encrypt %ld\n", len);
54     return new_sync_write(filp, buf, len, ppos);
55 }
56
57 ssize_t new_sync_read_crypt(struct file *filp, char __user *buf, size_t len, loff_t *ppos)
58 {
59     char* mybuf = (char*)kmalloc(sizeof(char)*len,GFP_KERNEL);
60     int i;
61     ssize_t ret = new_sync_read(filp, buf, len, ppos);
62     copy_from_user(mybuf,buf,len);
63     for(i = 0; i < len; i++){
64         mybuf[i] = ((mybuf[i] ^ 127) - 25 + 128) % 128;
65     }
66     copy_to_user(buf,mybuf,len);
67     printk("haha decrypt %ld\n", len);
68     return ret;
69 }
```

这里我使用了位移+异或的加密算法，解密时先异或再位移。

- 直接用 make 编译的时候产生以下问题:

```
jsb@ubuntu: /usr/src/linux/fs/myext2
jsb@ubuntu:/usr/src/linux/fs/myext2$ sudo 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
CC [M] /usr/src/linux-4.6/fs/myext2/dir.o
CC [M] /usr/src/linux-4.6/fs/myext2/file.o
/usr/src/linux-4.6/fs/myext2/file.c: In function 'new_sync_write_crypt':
/usr/src/linux-4.6/fs/myext2/file.c:54:9: error: implicit declaration of function 'new_sync_write' [-Werror=implicit-function-declaration]
    return new_sync_write(filp, buf, len, ppos);
           ^
/usr/src/linux-4.6/fs/myext2/file.c: In function 'new_sync_read_crypt':
/usr/src/linux-4.6/fs/myext2/file.c:61:23: error: implicit declaration of function 'new_sync_read' [-Werror=implicit-function-declaration]
    ssize_t ret = new_sync_read(filp, buf, len, ppos);
                      ^
cc1: some warnings being treated as errors
scripts/Makefile.build:291: recipe for target '/usr/src/linux-4.6/fs/myext2/file.o' failed
make[2]: *** [/usr/src/linux-4.6/fs/myext2/file.o] Error 1
Makefile:1428: recipe for target '_module_/usr/src/linux-4.6/fs/myext2' failed
make[1]: *** [_module_/usr/src/linux-4.6/fs/myext2] Error 2
make[1]: Leaving directory '/usr/src/linux-4.6'
Makefile:11: recipe for target 'default' failed
make: *** [default] Error 2
jsb@ubuntu:/usr/src/linux/fs/myext2$
```

- 排查了一下问题: 实验指导里的代码在 return 的时候调用了 new_sync_write() 和 new_sync_read() 函数, 而在我这个版本的 file.c 里并没有这两个函数的函数体。
- 通过查阅资料发现, 这两个函数的主体在 fs 文件夹下 read_write.c 里。我操作了半天也不知道该加什么头文件, 能把 fs/read_write.c 里的函数链接进 file.c。但是如果我没有这两个函数, 我的加密解密程序就无法正常运行……
- 最后我采用简单暴力的解决方案: 直接把这两个函数复制到 file.c 里。如下:

```
*file.c
/usr/src/linux-4.6/fs/myext2
Save
43 static ssize_t new_sync_read(struct file *filp, char __user *buf, size_t len,
44                             loff_t *ppos)
45 {
46     struct iovec iov = { .iov_base = buf, .iov_len = len };
47     struct kiocb kiocb;
48     struct iov_iter iter;
49     ssize_t ret;
50     init_sync_kiocb(&kiocb, filp);
51     kiocb.ki_pos = *ppos;
52     iov_iter_init(&iter, READ, &iov, 1, len);
53     ret = filp->f_op->read_iter(&kiocb, &iter);
54     BUG_ON(ret == -EIOCBQUEUED);
55     *ppos = kiocb.ki_pos;
56     return ret;
57 }
58
59
60 static ssize_t new_sync_write(struct file *filp, const char __user *buf, size_t
61                             len, loff_t *ppos)
62 {
63     struct iovec iov = { .iov_base = (void __user *)buf, .iov_len = len };
64     struct kiocb kiocb;
65     struct iov_iter iter;
66     ssize_t ret;
67     init_sync_kiocb(&kiocb, filp);
68     kiocb.ki_pos = *ppos;
69     iov_iter_init(&iter, WRITE, &iov, 1, len);
70 }
```

- 这个解决方案还是有问题：我在 make 的时候有以下报错：

```
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo 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/file.o
/usr/src/linux-4.6/fs/myext2/file.c: In function 'new_sync_read':
/usr/src/linux-4.6/fs/myext2/file.c:45:9: error: variable 'iov' has initializer but incomplete type
  struct iovec iov = { .iov_base = buf, .iov_len = len };
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:45:9: error: unknown field 'iov_base' specified in initializer
  struct iovec iov = { .iov_base = buf, .iov_len = len };
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:45:35: note: (near initialization for 'iov')
/usr/src/linux-4.6/fs/myext2/file.c:45:9: error: unknown field 'iov_len' specified in initializer
  struct iovec iov = { .iov_base = buf, .iov_len = len };
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:45:51: warning: excess elements in struct initializer
  struct iovec iov = { .iov_base = buf, .iov_len = len };
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:45:51: note: (near initialization for 'iov')
/usr/src/linux-4.6/fs/myext2/file.c:45:15: error: storage size of 'iov' isn't known
  struct iovec iov = { .iov_base = buf, .iov_len = len };
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:47:18: error: storage size of 'iter' isn't known
  struct iov_iter iter;
                          ^
/usr/src/linux-4.6/fs/myext2/file.c:52:2: error: implicit declaration of function 'iov_iter_init' [-Werror=implicit-func
declaration]
  iov_iter_init(&iter, READ, &iov, 1, len);
  ^
```

- 根据错误信息，程序不知道 iov 和 iov_base 的定义。我上网搜了搜，发现这些信息是定义在 uio.h 里的。最后我在 file.c 里加了一行终于编译成功了：

```
#include <linux/uio.h>
```

更新后的头文件信息如图：

```
21 #include <linux/time.h>
22 #include <linux/pagemap.h>
23 #include <linux/dax.h>
24 #include <linux/quotaops.h>
25 #include <linux/uio.h>
26 #include "myext2.h"
27 #include "xattr.h"
28 #include "acl.h"
29
30 #include <linux/gfp.h>
31
```

- 用 make 重新编译 myext2 模块，用命令 insmod 安装编译好的 myext2.ko 内核模块：

```
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo 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
CC [M] /usr/src/linux-4.6/fs/myext2/dir.o
CC [M] /usr/src/linux-4.6/fs/myext2/file.o
CC [M] /usr/src/linux-4.6/fs/myext2/ialloc.o
CC [M] /usr/src/linux-4.6/fs/myext2/inode.o
CC [M] /usr/src/linux-4.6/fs/myext2/ioctl.o
CC [M] /usr/src/linux-4.6/fs/myext2/namei.o
CC [M] /usr/src/linux-4.6/fs/myext2/super.o
CC [M] /usr/src/linux-4.6/fs/myext2/symlink.o
LD [M] /usr/src/linux-4.6/fs/myext2/myext2.o
Building modules, stage 2.
MODPOST 1 modules
CC /usr/src/linux-4.6/fs/myext2/myext2.mod.o
LD [M] /usr/src/linux-4.6/fs/myext2/myext2.ko
make[1]: Leaving directory '/usr/src/linux-4.6'
jsb@ubuntu: /usr/src/linux/fs/myext2$ sudo insmod myext2.ko
```

- 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/
```

五、实验结果和分析

5.1 加密解密的测试结果

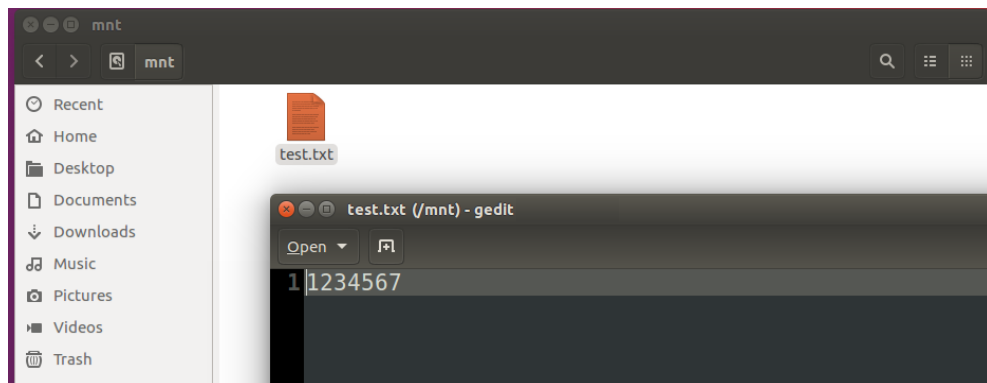
- 在 /mnt 里新建一个文件 test.txt（注意要用 root 权限）

```
jsb@ubuntu:/mnt$ touch test.txt
touch: cannot touch 'test.txt': Permission denied
jsb@ubuntu:/mnt$ sudo touch test.txt
```

- 在 /mnt 里用 root 权限打开 gedit，输入 1234567 并保存：

```
jsb@ubuntu:/mnt$ sudo gedit test.txt

(gedit:3026): IBUS-WARNING **: The owner of /home/jsb/.confi
(gedit:3026): Gtk-WARNING **: Calling Inhibit failed: GDBus
erviceUnknown: The name org.gnome.SessionManager was not pro
** (gedit:3026): WARNING **: Set document metadata failed:
ell-enabled not supported
```



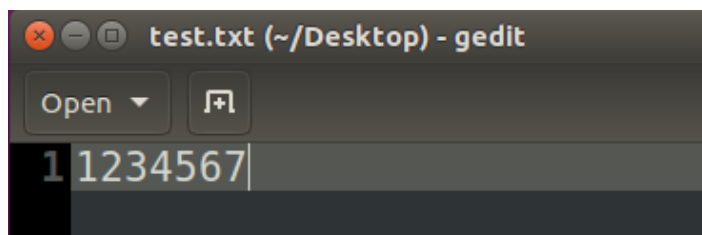
- 在 /mnt 里用 touch 命令查看文件，发现被正确地解密：

```
jsb@ubuntu:/mnt$ cat test.txt
1234567
```

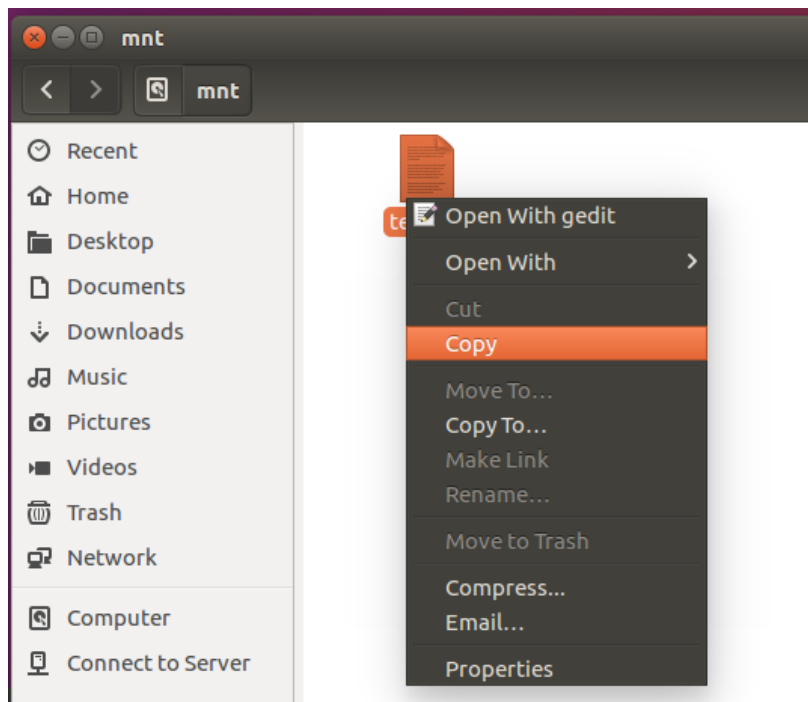
- 用 cp 命令把该 test.txt 复制到桌面：

```
jsb@ubuntu:/mnt$ cp test.txt /home/jsb/Desktop/
```

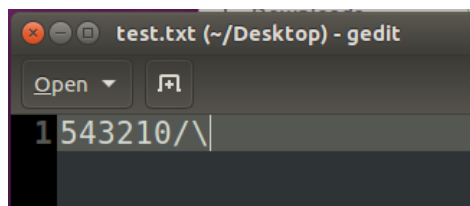
- 在桌面用 gedit 打开 test.txt，发现被正确地解密：



- 用文件管理器（我的理解是用 Linux 的）把 test.txt 复制到桌面：



- 此时打开 test.txt，发现没有被解密（看到的是密文）：



计算一下，0 的 ASCII 码是 48， $(48+25)^{127}=54$ ，正好对应数字 5，即加密正确。

- 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
```

- 即使是在 /mnt 里，执行 cat 后得到的依然是密文：

```
jsb@ubuntu:/mnt$ cat test.txt
543210/\
```


六、讨论心得

1. 本次实验做的是文件系统，核心是 `mount` 这个操作。
2. 相比于之前两个实验，本次实验我的手脚“利索”了很多，在控制台敲命令也可谓“熟门熟路”了。尽管我还是遇到了不少棘手的问题（**为了叙述的连贯和流畅，具体问题和解决方案已经在上两节里详细讨论**），但现在的心态明显好了很多，遇到错误可以冷静地分析问题，最终稳健地解决。
3. 我觉得 Linux 的文件系统还是蛮神奇、蛮有趣的。通过简单的挂载命令，可以把一块空间快速连上一个文件系统。就拿本实验来说吧，实现好了这个内核模块后，写入装载谋爱、`mount` 的命令，`\mnt` 目录下就立刻挂载了该文件系统。通过该文件系统，**当`\mnt`的文件与外界产生交互时，表现出来的就是密文的形式**，特别有趣。
4. “纸上得来终觉浅”，光看着实验要求里那几行看似简单的命令一定是没有收获的；必须自己实践过才能出真知。