**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： 专业： 自动化 学号：

电子邮件地址： 手机：

实验地点： 玉泉曹光彪2期503 实验日期： 2019 年 12 月 24 日

**一、实验目的**

1. 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。

2. 深入理解操作系统文件系统原理

3. 学习理解Linux的VFS文件系统管理技术

4. 学习理解Linux的ext2文件系统实现技术

5. 设计和实现加密文件系统

1. **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：在linux操作系统环境下重建内核

* 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**

1. VMware workstation pro

2. Ubuntu16.04 LTE

3. Intel core i5-7300HQ

4. 虚拟机RAM：2G

**四、操作方法与实验步骤**

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myext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：

1. #cd /usr/src/linux  /\*kernel source code\*/
2. #cd fs
3. #cp -R ext2 myext2
4. #cd /usr/src/linux/fs/myext2
5. #mv ext2.h myext2.h
7. #cd /lib/modules/$(uname -r)/build/include/linux
8. #cp ext2\_fs.h myext2\_fs.h
9. #cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops
10. #cp ext2-atomic.h myext2-atomic.h
11. #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：

1. #!/bin/bash
3. SCRIPT=substitute.sh
5. **for** f in \*
6. **do**
7. **if** [ $f = $SCRIPT ]
8. then
9. echo "skip $f"
10. **continue**
11. fi
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"
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"
23. 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vim编辑器中的具体操作为：

1. :%s/ext2/myext2/g
2. :%s/EXT2/MYEXT2/g

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

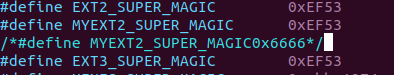
#include <asm-generic/bitops/myext2-atomic.h>

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

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

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

#define MYEXT2\_SUPER\_MAGIC 0xEF53



源代码的修改工作到此结束。接下来就是第三步工作—一把myext2编译源成内核模块。 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：

1. #
2. # Makefile for the linux myext2-filesystem routines.
3. #
4. obj-m := myext2.o
5. myext2-y := balloc.o dir.o file.o ialloc.o inode.o \
6. ioctl.o namei.o super.o symlink.o
8. KDIR := /lib/modules/$(shell uname -r)/build
9. PWD := $(shell pwd)
10. **default**:
11. make -C $(KDIR) M=$(PWD) modules
12. clean:
13. make -C $(KDIR) M=$(PWD) clean

编译好模块后，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：

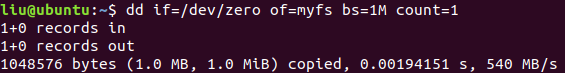
1. insmod myext2.ko
2. cat /proc/filesystems | grep myext2

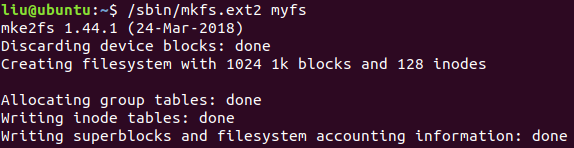


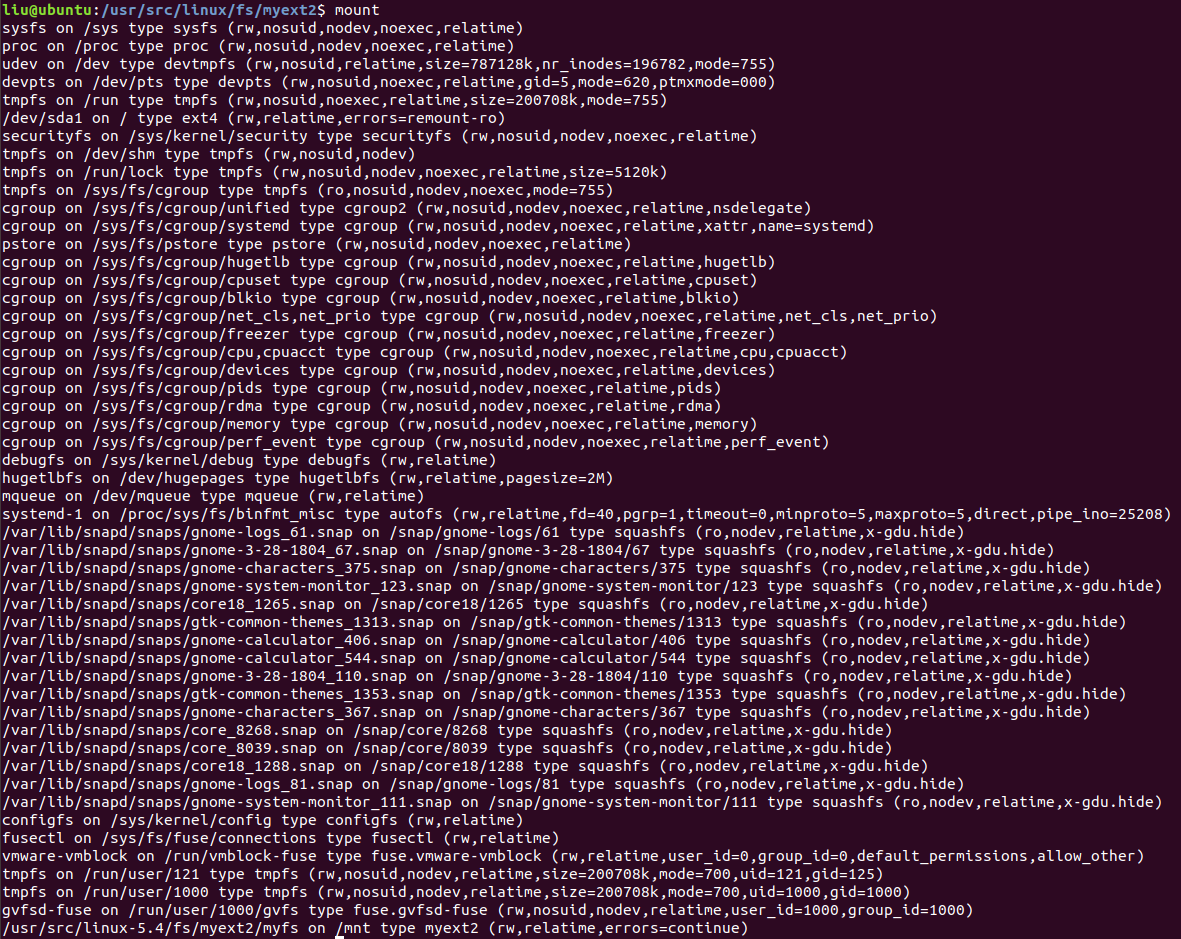
确认myext2文件系统加载成功后，可以对添加的myext2文件系统进行测试了，输入命令cd先把当前目录设置成主目录。

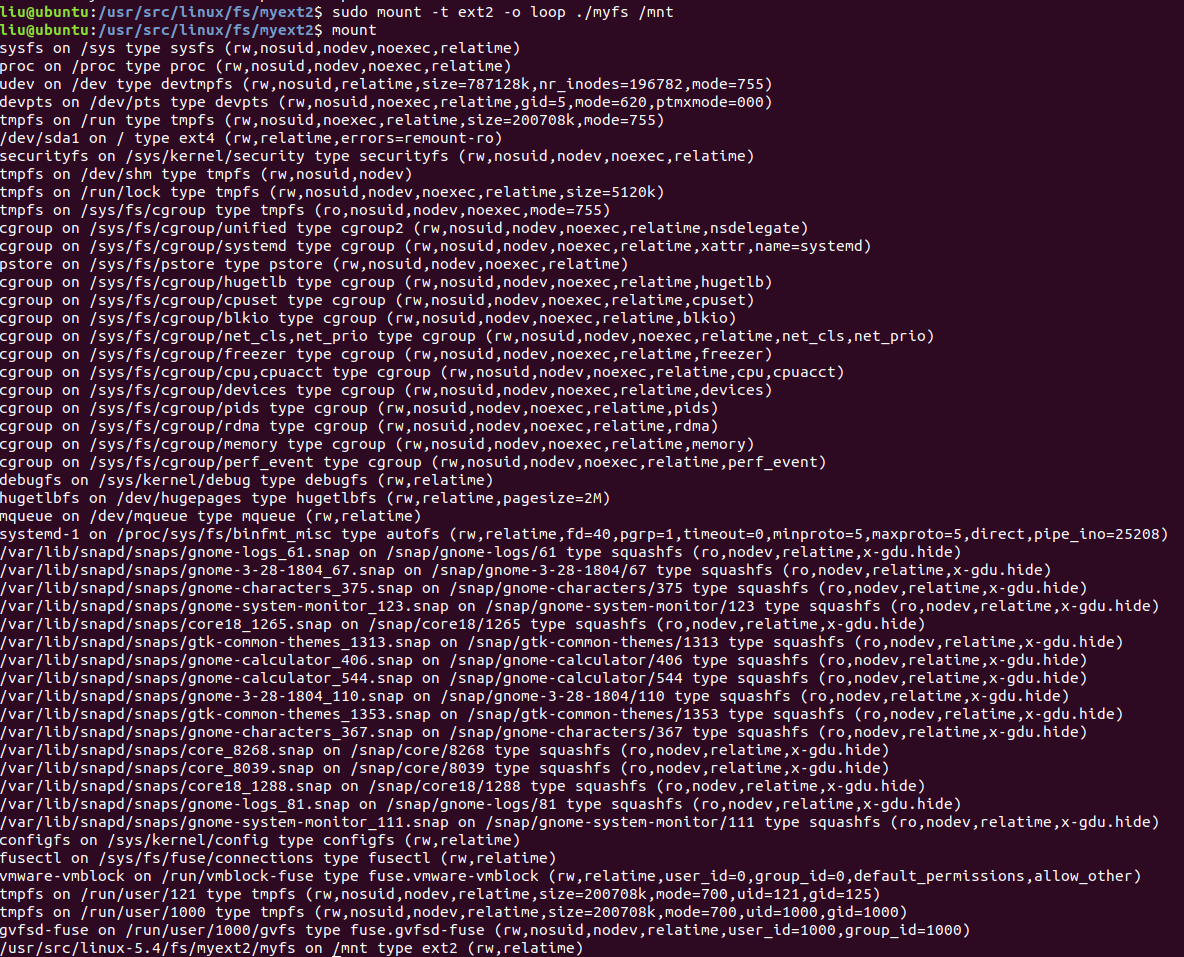
对添加的myext2文件系统测试命令如下：

1. #dd if=/dev/zero of=myfs bs=1M count=1
2. #/sbin/mkfs.ext2 myfs
3. #mount -t myext2 -o loop ./myfs /mnt
4. #mount
5. #umount /mnt
6. #mount -t ext2 -o loop ./myfs /mnt
7. #mount
8. #umount /mnt
9. #rmmod myext2









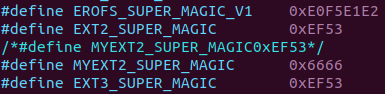
2．修改myext2的magic number

在上面做的基础上。找到myext2的magic number，并将其改为0x6666：

3.18.24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



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。

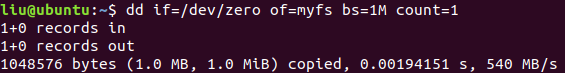
changeMN.c程序可以在课程网站中下载。这个程序经过编译后产生的可执行程序名字为changeMN。

changeMN.c:

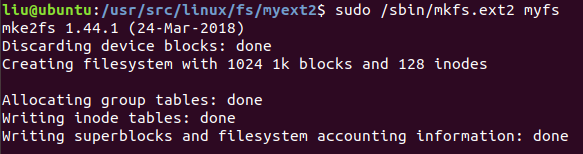
1. #include <stdio.h>
2. **int** main()
3. {
4. **int** ret;
5. **FILE** \*fp\_read;
6. **FILE** \*fp\_write;
7. unsigned **char** buf[2048];
9. fp\_read = fopen("./myfs", "rb");
11. **if** (fp\_read == NULL)
12. {
13. printf("open myfs failed!\n");
14. **return** 1;
15. }
17. fp\_write = fopen("./newfs", "wb");
19. **if** (fp\_write == NULL)
20. {
21. printf("open newfs failed!\n");
22. **return** 2;
23. }
25. ret = fread(buf, **sizeof**(unsigned **char**), 2048, fp\_read);
27. printf("previous magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
29. buf[0x438] = 0x66;
30. buf[0x439] = 0x66;
32. fwrite(buf, **sizeof**(unsigned **char**), 2048, fp\_write);
34. printf("current magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
36. **while** (ret == 2048)
37. {
38. ret = fread(buf, **sizeof**(unsigned **char**), 2048, fp\_read);
39. fwrite(buf, **sizeof**(unsigned **char**), ret, fp\_write);
40. }
42. **if** (ret < 2048 && feof(fp\_read))
43. {
44. printf("change magic number ok!\n");
45. }
46. fclose(fp\_read);
47. fclose(fp\_write);
49. **return** 0;
50. }

下面我们开始测试：

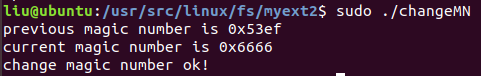
#dd if=/dev/zero of=myfs bs=1M count=1



#/sbin/mkfs.ext2 myfs

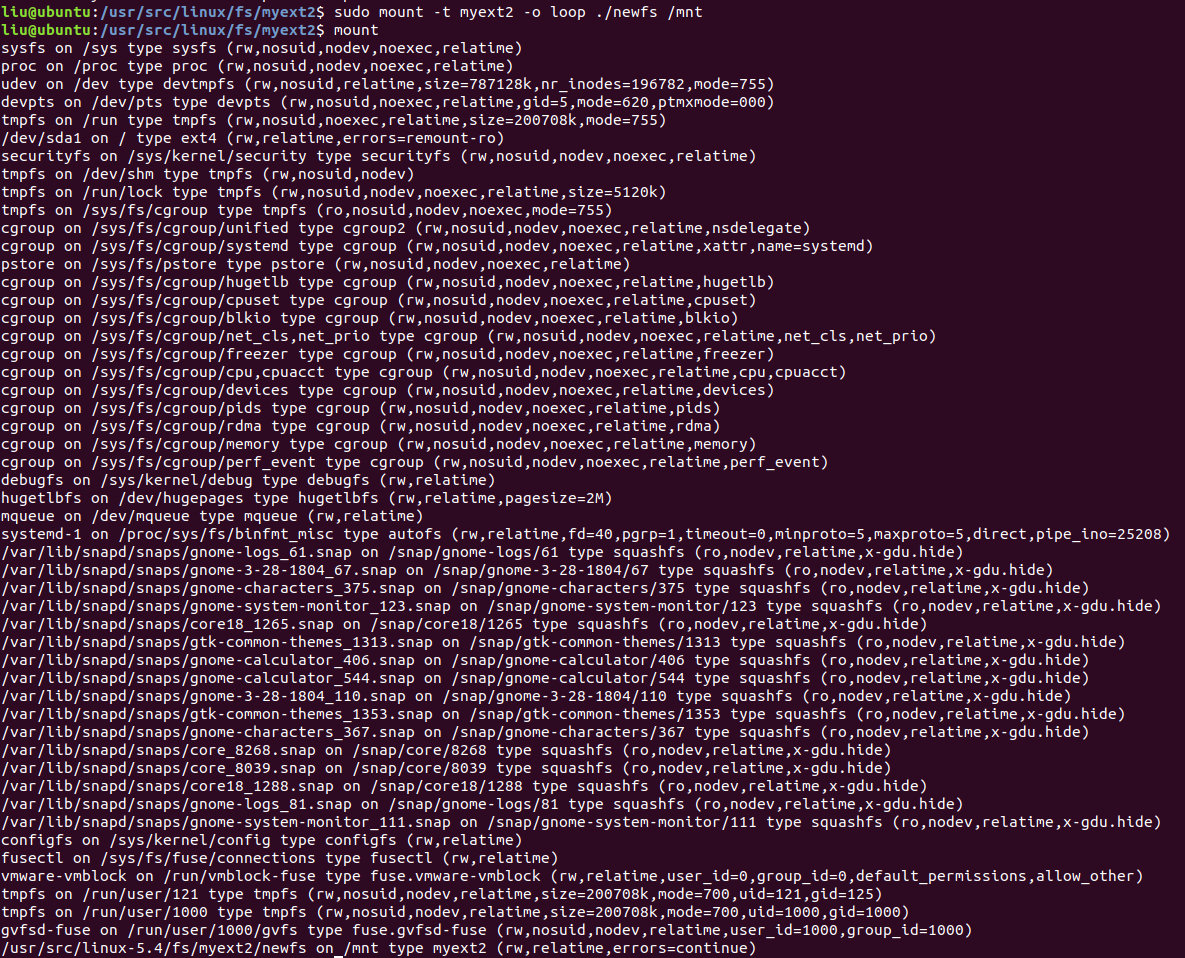


#./changeMN



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

#mount



#sudo umount /mnt

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



# rmmod myext2

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 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

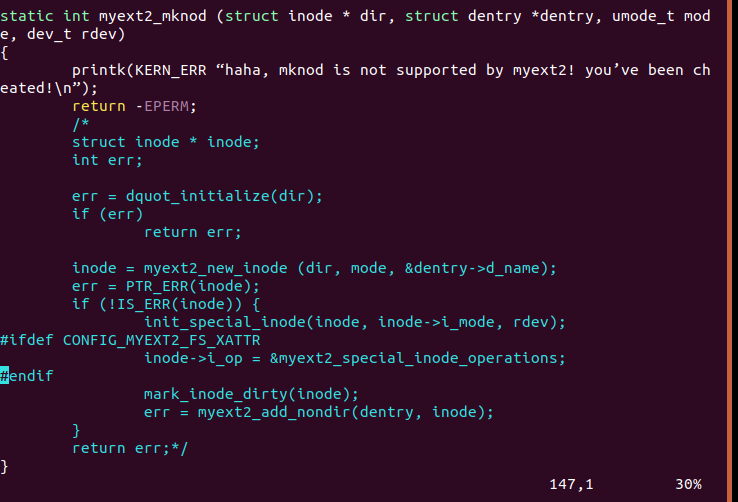
408 .setattr = ext2\_setattr,

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



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，maknod不被支持。

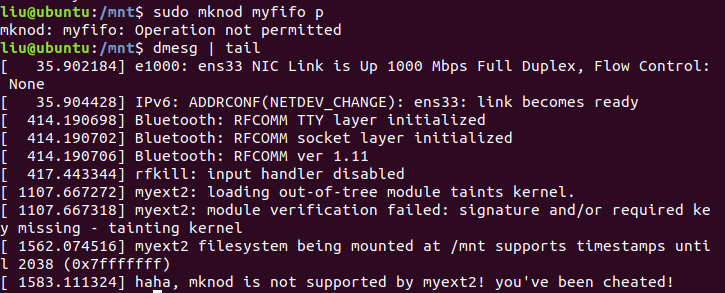
修改完毕，再用make重新编译内核模块，使用命令insmod安装编译好的myext2.ko内核模块。我们在shell下执行如下测试程序：

1. #mount -t myext2 -o loop ./fs.new /mnt
2. #cd /mnt
3. #mknod myfifo p
4. #mount -t myext2 -o loop ./fs.new /mnt
5. #cd /mnt
6. #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。



可见，我们的裁减工作取得了预期的效果。

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

./Newchange

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newfs，并且将newfs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。

Newchange.c:

1. #include <stdio.h>
2. **int** main()
3. {
4. **int** ret;
5. **FILE** \*fp\_read;
6. **FILE** \*fp\_write;
7. unsigned **char** buf[2048];
9. fp\_read = fopen("./tmpfs", "rb");
11. **if** (fp\_read == NULL)
12. {
13. printf("open tmpfs failed!\n");
14. **return** 1;
15. }
17. fp\_write = fopen("./newfs", "wb");
19. **if** (fp\_write == NULL)
20. {
21. printf("open newfs failed!\n");
22. **return** 2;
23. }
25. ret = fread(buf, **sizeof**(unsigned **char**), 2048, fp\_read);
27. printf("previous magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
29. buf[0x438] = 0x66;
30. buf[0x439] = 0x66;
32. fwrite(buf, **sizeof**(unsigned **char**), 2048, fp\_write);
34. printf("current magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
36. **while** (ret == 2048)
37. {
38. ret = fread(buf, **sizeof**(unsigned **char**), 2048, fp\_read);
39. fwrite(buf, **sizeof**(unsigned **char**), ret, fp\_write);
40. }
42. **if** (ret < 2048 && feof(fp\_read))
43. {
44. printf("change magic number ok!\n");
45. }
46. fclose(fp\_read);
47. fclose(fp\_write);
49. **return** 0;
50. }

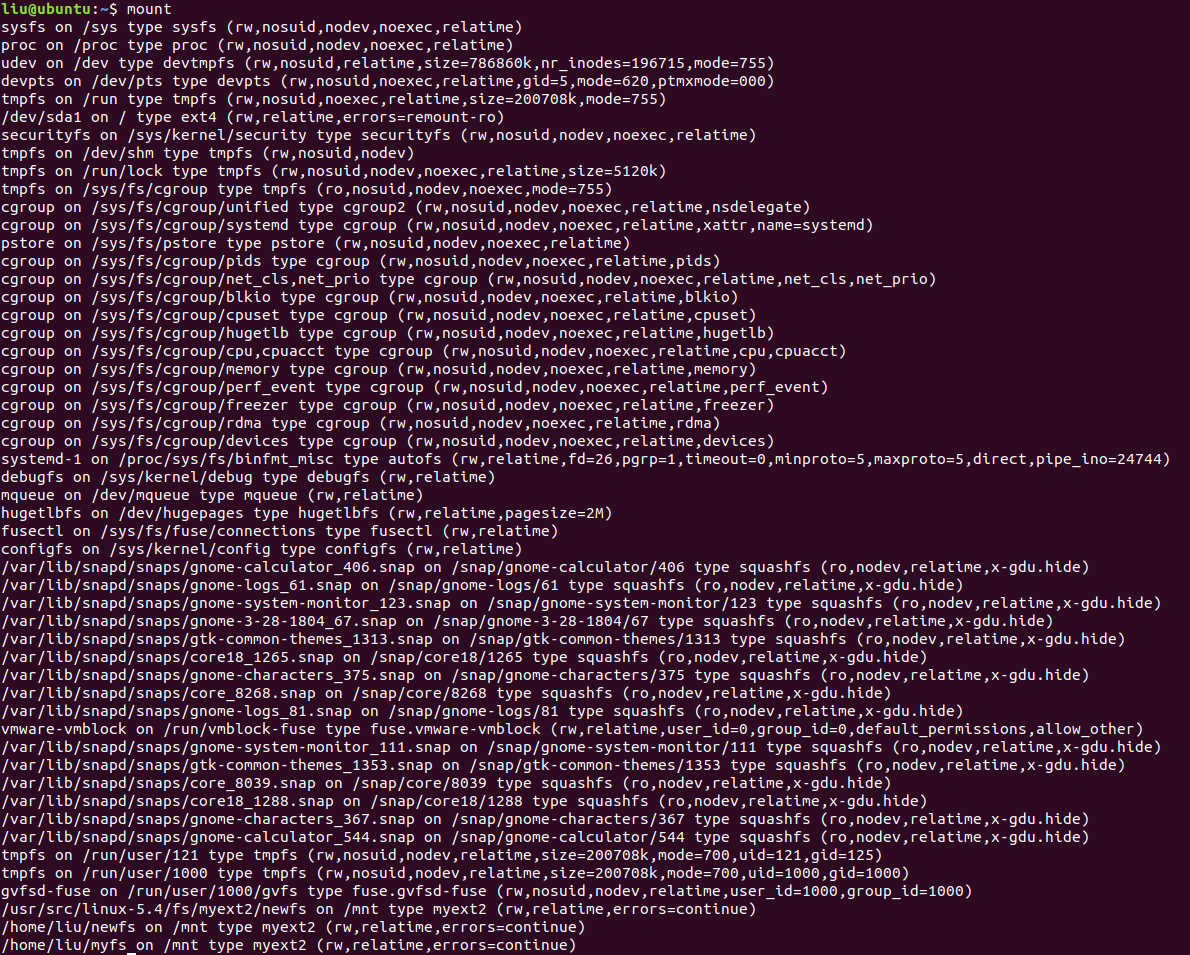
编辑完了之后，做如下测试：

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

# sudo bash mkfs.myext2 myfs

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

# mount



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。改动部分如下所示。

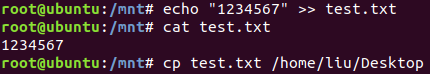
1. #include <linux/uio.h>/\*added at the top\*/
2. /\*
3. ...
4. \*/
5. #include "acl.h"
6. **static** ssize\_t new\_sync\_read(**struct** file \*filp, **char** \_\_user \*buf, **size\_t** len, loff\_t \*ppos)
7. {
8. **struct** iovec iov = { .iov\_base = buf, .iov\_len = len };
9. **struct** kiocb kiocb;
10. **struct** iov\_iter iter;
11. ssize\_t ret;
13. init\_sync\_kiocb(&kiocb, filp);
14. kiocb.ki\_pos = \*ppos;
15. iov\_iter\_init(&iter, READ, &iov, 1, len);
17. ret = filp->f\_op->read\_iter(&kiocb, &iter);
18. BUG\_ON(ret == -EIOCBQUEUED);
19. \*ppos = kiocb.ki\_pos;
20. **return** ret;
21. }
23. **static** ssize\_t new\_sync\_write(**struct** file \*filp, **const** **char** \_\_user \*buf, **size\_t** len, loff\_t \*ppos)
24. {
25. **struct** iovec iov = { .iov\_base = (**void** \_\_user \*)buf, .iov\_len = len };
26. **struct** kiocb kiocb;
27. **struct** iov\_iter iter;
28. ssize\_t ret;
30. init\_sync\_kiocb(&kiocb, filp);
31. kiocb.ki\_pos = \*ppos;
32. iov\_iter\_init(&iter, WRITE, &iov, 1, len);
34. ret = filp->f\_op->write\_iter(&kiocb, &iter);
35. BUG\_ON(ret == -EIOCBQUEUED);
36. **if** (ret > 0)
37. \*ppos = kiocb.ki\_pos;
38. **return** ret;
39. }
40. **static** ssize\_t new\_sync\_read\_crypt(**struct** file \*filp,**char** \_\_user \*buf, **size\_t** len,loff\_t \*ppos)
41. {
42. **int** i;
43. **char**\* mybuf=(**char**\*)kmalloc(**sizeof**(**char**)\*len,GFP\_KERNEL);
44. ssize\_t ret=new\_sync\_read(filp,buf,len,ppos);
45. copy\_from\_user(mybuf,buf,len);
46. **for**(i=0;i<len;++i)mybuf[i]=(mybuf[i]-25+128)%128;
47. copy\_to\_user(buf,mybuf,len);
48. printk("haha decrypt %u\n",len);
49. **return** ret;
50. }
52. **static** ssize\_t new\_sync\_write\_crypt(**struct** file \*filp,**const** **char** \_\_user \*buf,**size\_t** len, loff\_t \*ppos)
53. {
54. **int** i;
55. **char**\* mybuf=(**char**\*)kmalloc(**sizeof**(**char**)\*len,GFP\_KERNEL);
56. copy\_from\_user(mybuf,buf,len);
57. **for**(i=0;i<len;++i)mybuf[i]=(mybuf[i]+25)%128;
58. copy\_to\_user(buf,mybuf,len);
59. printk("haha encrypt %u\n",len);
60. **return** new\_sync\_write(filp,buf,len,ppos);
61. }
62. /\*
63. ...
64. \*/
66. **const** **struct** file\_operations myext2\_file\_operations = {
67. .read           = new\_sync\_read\_crypt,
68. .write          = new\_sync\_write\_crypt,
69. .llseek         = generic\_file\_llseek,
70. .read\_iter      = generic\_file\_read\_iter,
71. .write\_iter     = generic\_file\_write\_iter,
72. .unlocked\_ioctl = myext2\_ioctl,
73. #ifdef CONFIG\_COMPAT
74. .compat\_ioctl   = myext2\_compat\_ioctl,
75. #endif
76. .mmap           = myext2\_file\_mmap,
77. .open           = dquot\_file\_open,
78. .release        = myext2\_release\_file,
79. .fsync          = myext2\_fsync,
80. .splice\_read    = generic\_file\_splice\_read,
81. .splice\_write   = iter\_file\_splice\_write,
82. };
84. **const** **struct** inode\_operations myext2\_file\_inode\_operations = {
85. #ifdef CONFIG\_MYEXT2\_FS\_XATTR
86. .setxattr       = generic\_setxattr,
87. .getxattr       = generic\_getxattr,
88. .listxattr      = myext2\_listxattr,
89. .removexattr    = generic\_removexattr,
90. #endif
91. .setattr        = myext2\_setattr,
92. .get\_acl        = myext2\_get\_acl,
93. .set\_acl        = myext2\_set\_acl,
94. .fiemap         = myext2\_fiemap,
95. };

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。

1. mount -t myext2 -o loop ./newfs /mnt/
2. cd /mnt/

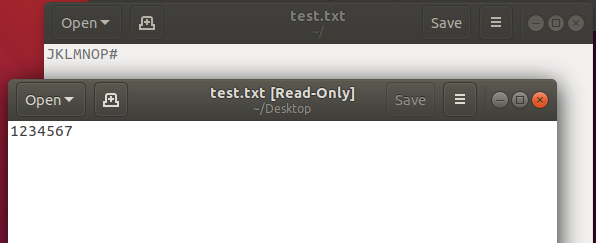
新建文件test.txt并写入字符串“1234567”，再查看test.txt文件内容：cat test.txt 。

把test.txt文件复制到主目录下：cp test.txt ~ 。

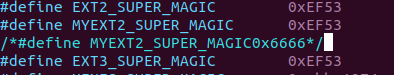


在主目录下打开test.txt文件，查看test.txt文件内容的结果

使用文件管理器的复制，再查看结果？



我们把之前的magic number（只需要改变include/uapi/linux/magic.h 文件中 #define部分number即可）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

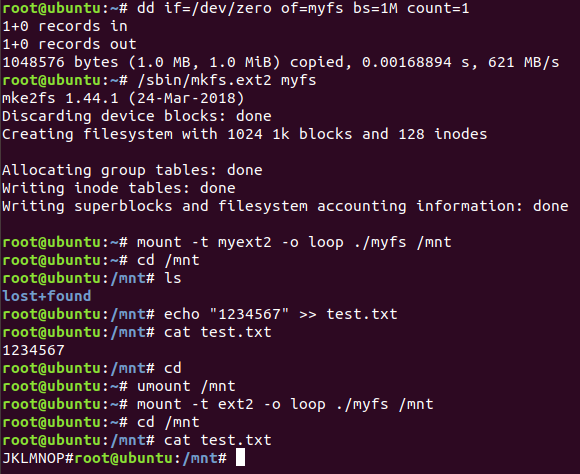
umount /mnt

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。



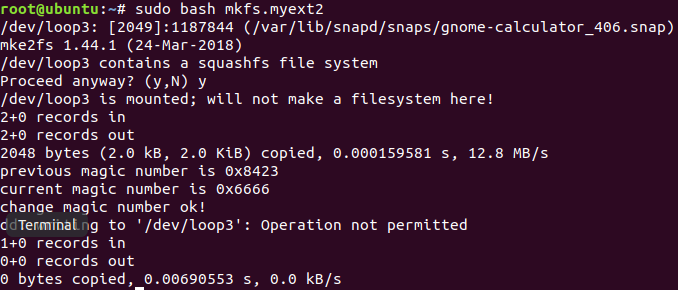
**五、讨论和心得**

**心得体会:**

在这个测试中，shell中的cp和GUI中的拷贝有一些不同。shell中的cp将首先读取文件中的数据，然后写回新位置的文件。因此，当我们从文件系统中复制文件时，myext2被挂载在/mnt中，而其他的fs在我的ubuntu上运行。我们先解密数据，然后写回，这样磁盘中的数据存储就不会加密。因此，在没有解密的情况下，原始文件系统中的读取操作可以显示原始数据。但是GUI中的副本有不同的结果，也许副本是使用mmap而不是读取。GUI中的副本不需要首先读取数据，它们直接映射数据。因此，使用原来的读取操作将得到加密结果。

遇到的问题：

1.在用脚本对loop3做操作时出现下图的问题：（先是对loop2做操作出现同样问题，所以改为对loop3做操作，同样出现问题）



最终的解决方法是：将loop2 umount下，然后再重复之前的操作即可。

2.在最后一步“把之前的magic number改回0xEF53” 只需要改变#define部分number即可，在编译时会直接将myfs中的文件mn改回去。我之前尝试使用了changeMN程序修改为Oxef56，但是每次修改后都会出现previous mn是ox56ef的情况。