**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： 添加一个加密文件系统

学生姓名： 张佳瑶 学号： 3170103240

电子邮件地址： 1531077171@qq.com

实验日期： 2019 年 12 月 17 日

1. 实验环境

处理器：Intel® Core™ i7-6700HQ CPU @ 2.60GHz

Windows10

Linux version 4.15.18 (zjy@ubuntu) (gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.11)) #5 SMP Sun Dec 15 00:27:02 PST 2019

1. 实验内容和结果及分析

实验设计思路：

获得实验二编译好的内核源码。在此基础上，修改Linux原有的文件系统中的代码，添加自己设计的加密读写程序，编译好后挂载到/mnt上，最终实现添加一个加密文件系统。

实验步骤及截图：

1. **添加一个类似ext2的文件系统myext2**

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的内核代码文件下，在 Linux 的 shell 下，执行如下操作:

#cd fs

#cp –R ext2 myext2

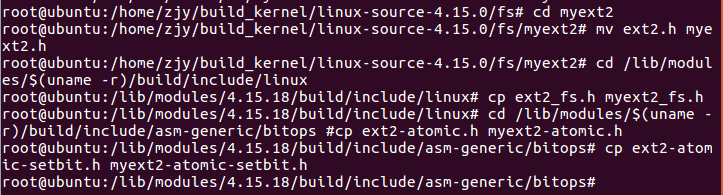
#cd /usr/src/linux/fs/myext2 #mv ext2.h myext2.h

#cd /lib/modules/$(uname -r)/build/include/linux

#cp ext2\_fs.h myext2\_fs.h

#cd /lib/modules/$(uname -r)/build/include/asm-generic/bitops #cp ext2-atomic.h myext2-atomic.h

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



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

#!/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

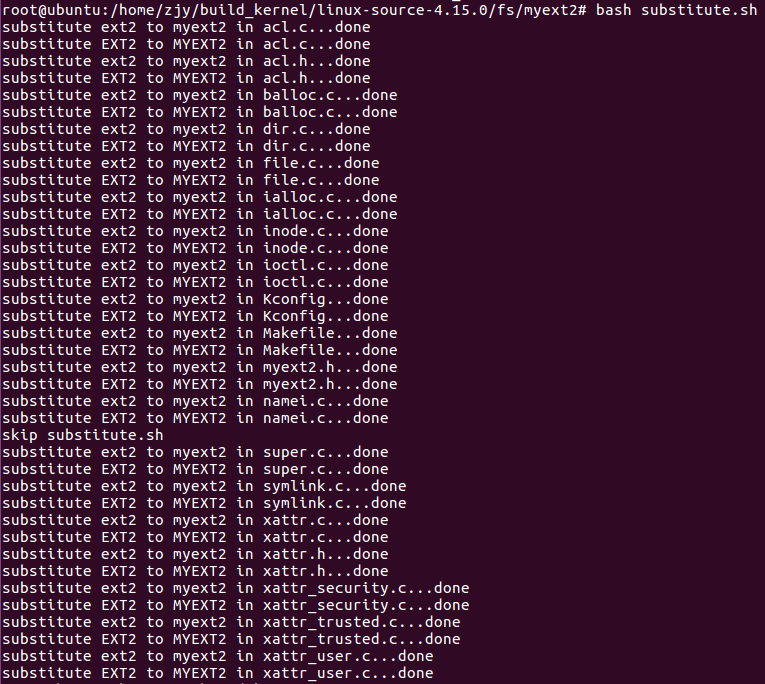
mv ${f}\_tmp $f

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”。

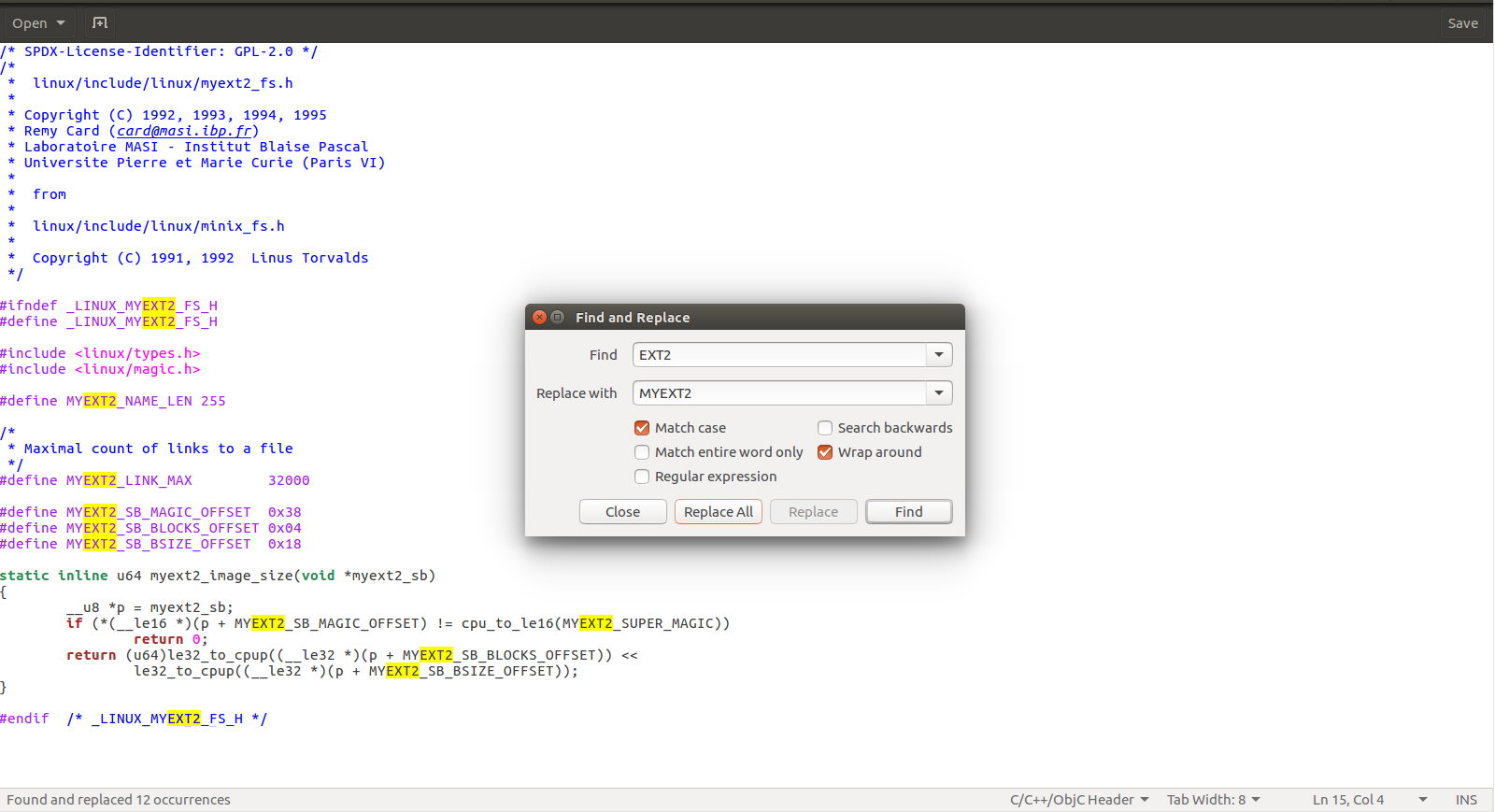
执行脚本：

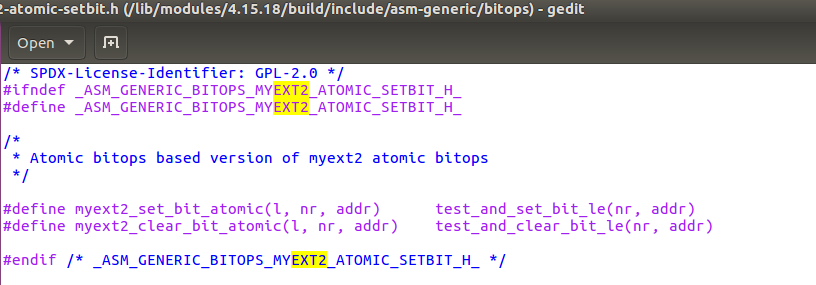


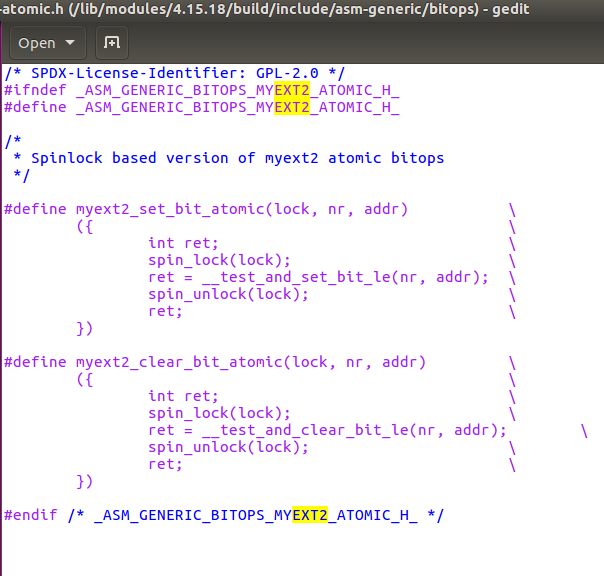
执行完成之后发现文件均变成只读文件，于是使用进行权限修改操作：



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

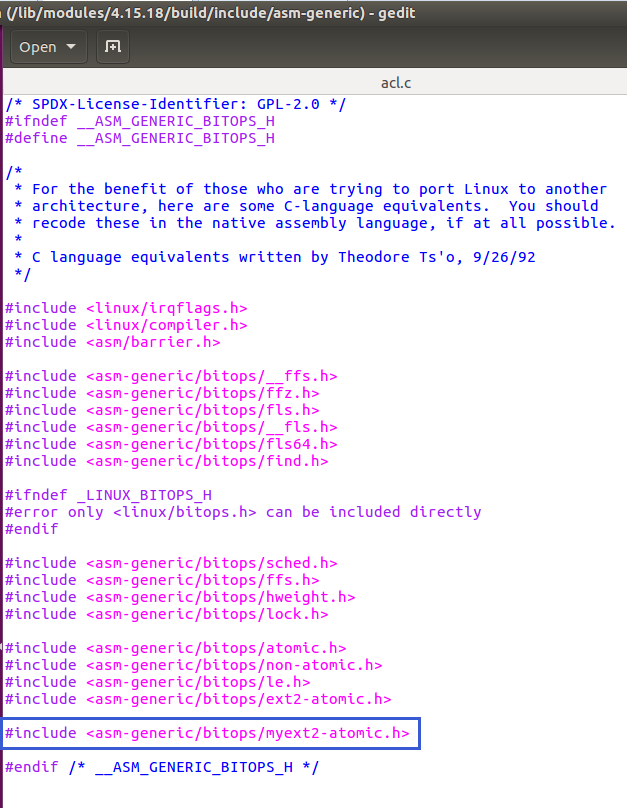






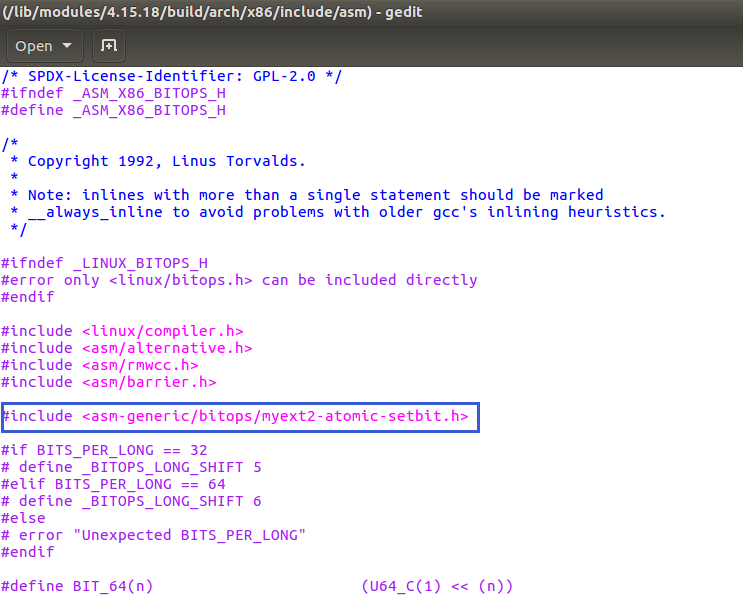
在/lib/modules/$(uname -r)/build /include/asm-generic/bitops.h文件中添加：

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

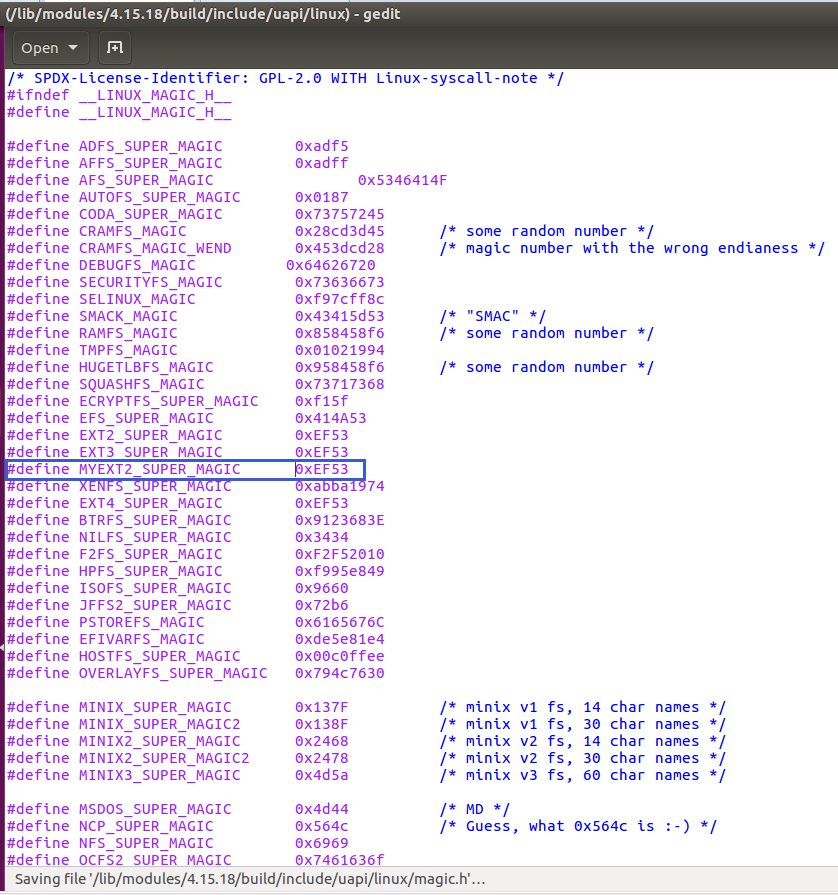


在/lib/modules/$(uname -r)/build /arch/x86/include/asm/bitops.h文件中添加：

#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/fm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 \

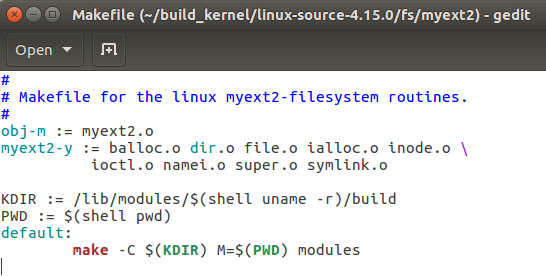
ioctl.o namei.o super.o symlink.o

KDIR := /lib/modules/$(shell uname -r)/build

PWD := $(shell pwd)

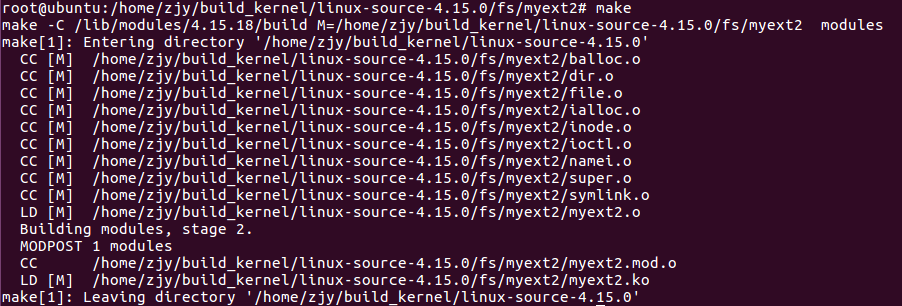
default:

make -C $(KDIR) M=$(PWD) modules



在myext2目录下执行命令：

#make



使用insmod命令加载模块：

#insmod myext2.ko

查看一下myext2文件系统是否加载成功：

#cat /proc/filesystems |grep myext2



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

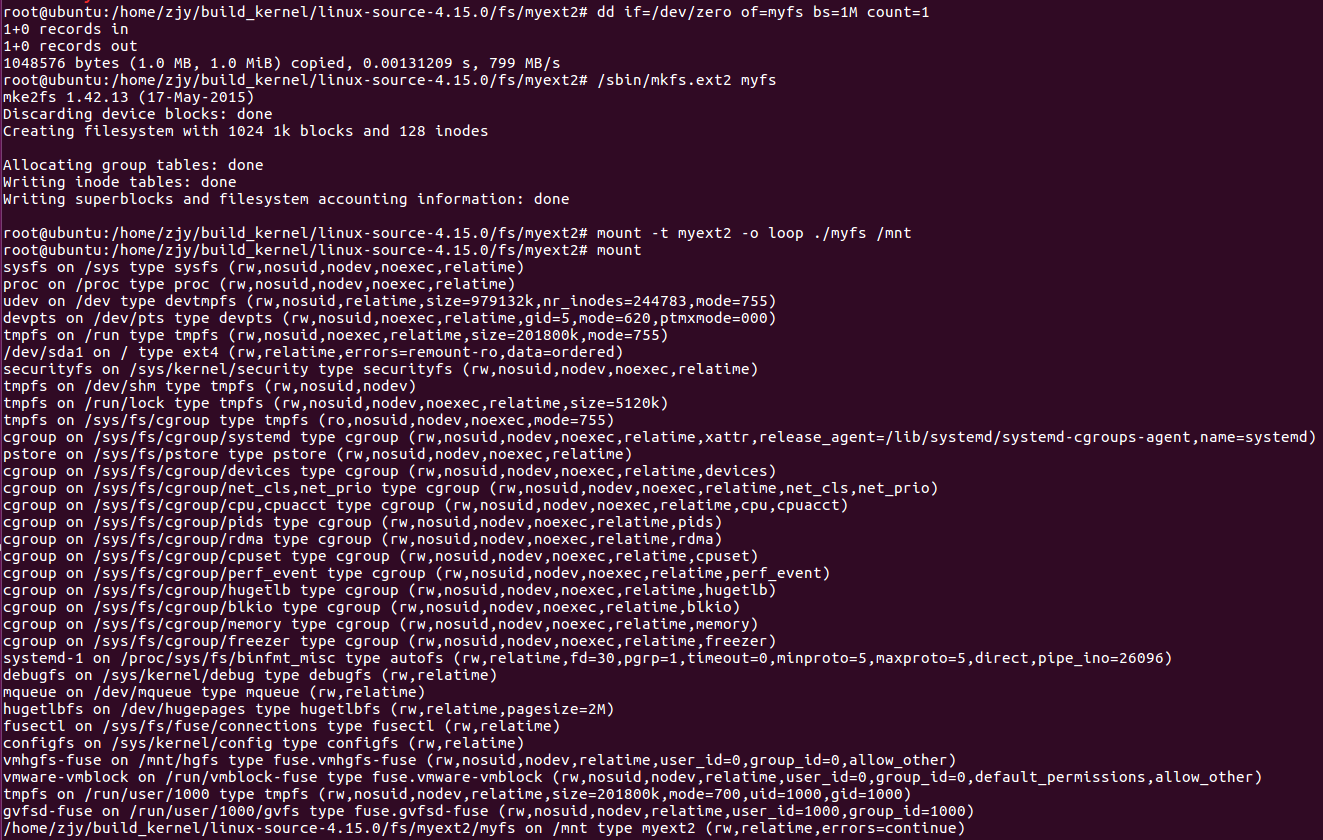
#mount

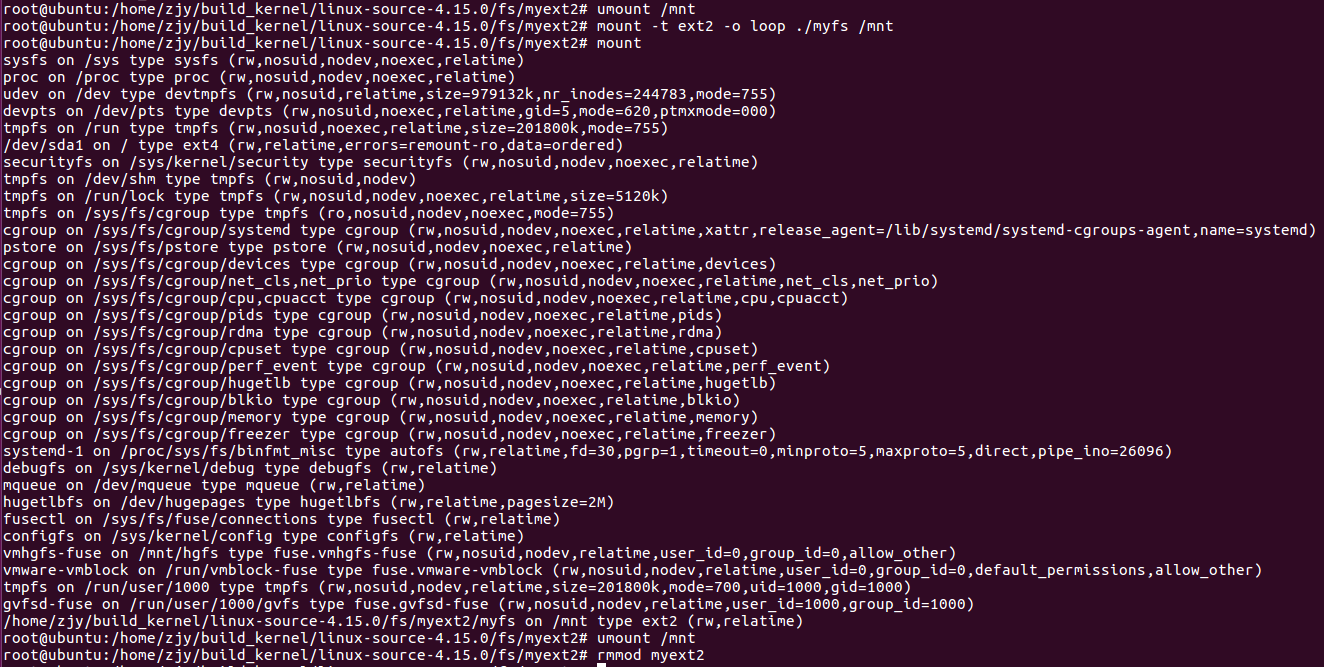
……

…… on /mnt type ext2 (rw)

#umount /mnt

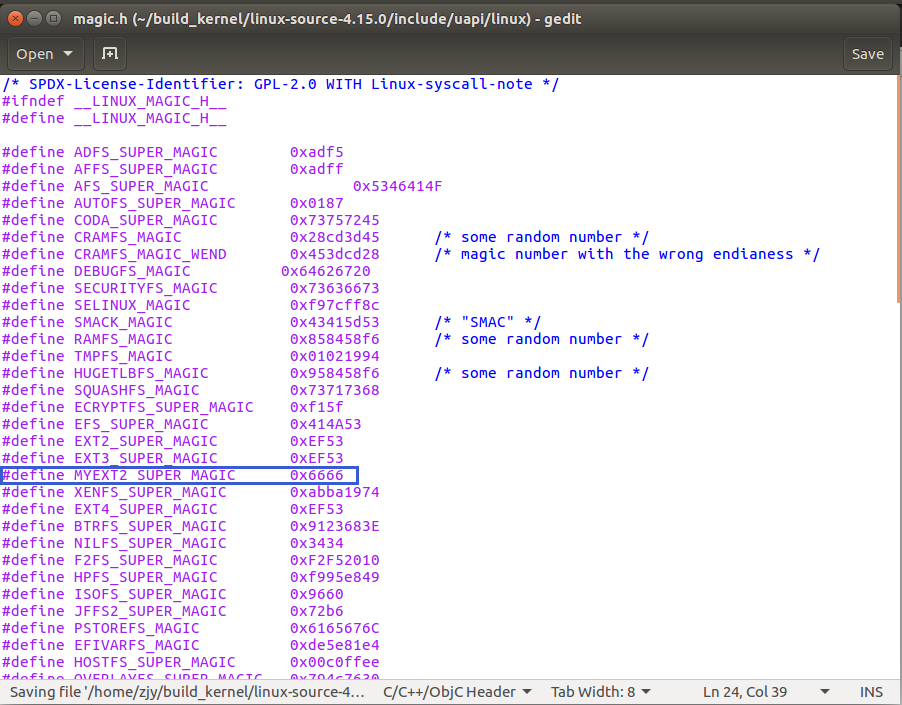
#rmmod myext2 /\*卸载模块\*/



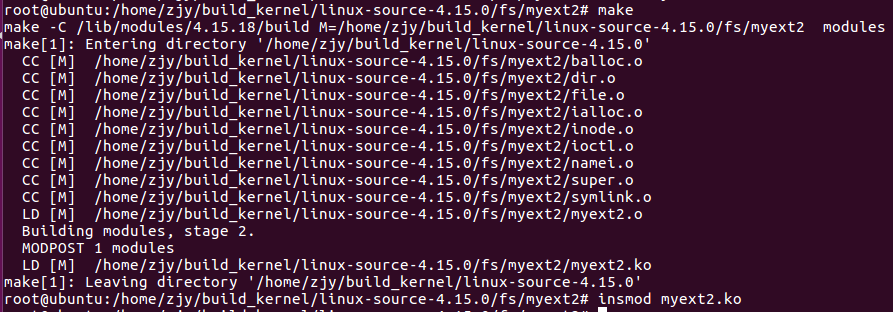


1. **修改myext2的magic number**

找到myext2的magic number，并将其改为0x6666



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。



编写changeMN.c:

#include <stdio.h>

main()

{

int ret;

FILE \*fp\_read;

FILE \*fp\_write;

unsigned char buf[2048];

fp\_read=fopen("./myfs","rb");

if(fp\_read == NULL)

{

printf("open myfs failed!\n");

return 1;

}

fp\_write=fopen("./fs.new","wb");

if(fp\_write==NULL)

{

printf("open fs.new failed!\n");

return 2;

}

ret=fread(buf,sizeof(unsigned char),2048,fp\_read);

printf("previous magic number is 0x%x%x\n",buf[0x438],buf[0x439]);

buf[0x438]=0x66;

buf[0x439]=0x66;

fwrite(buf,sizeof(unsigned char),2048,fp\_write);

printf("current magic number is 0x%x%x\n",buf[0x438],buf[0x439]);

while(ret == 2048)

{

ret=fread(buf,sizeof(unsigned char),2048,fp\_read);

fwrite(buf,sizeof(unsigned char),ret,fp\_write);

}

if(ret < 2048 && feof(fp\_read))

{

printf("change magic number ok!\n");

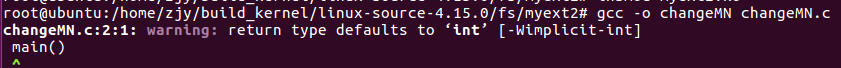
}

fclose(fp\_read);

fclose(fp\_write);

return 0;

}

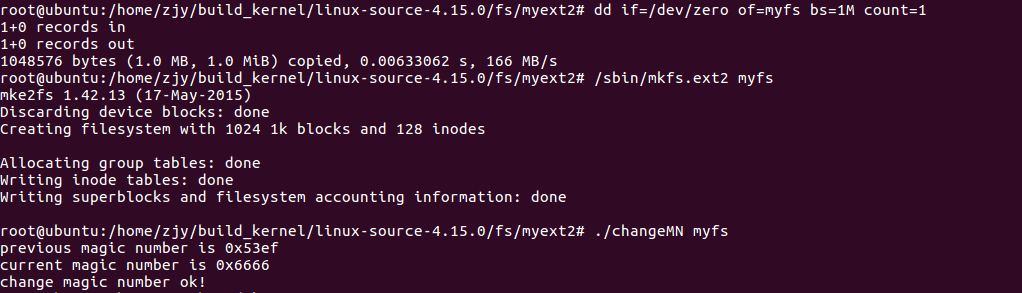


下面我们开始测试:

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

#/sbin/mkfs.ext2 myfs

#./changeMN myfs

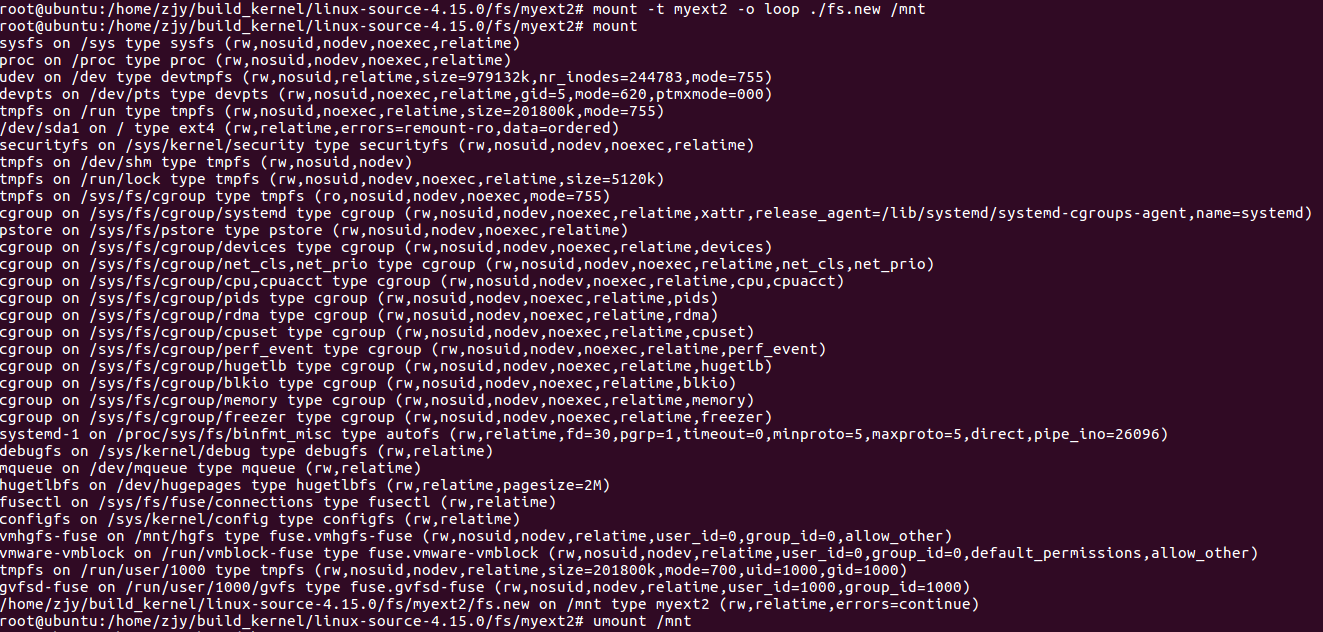


#mount -t myext2 -o loop ./fs.new /mnt

#mount

...... on /mnt type myext2 (rw)

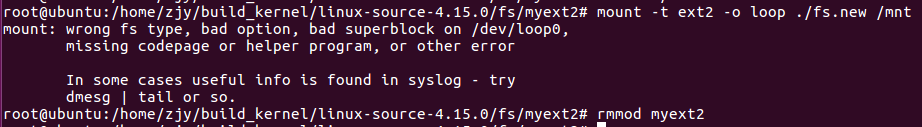
#sudo umount /mnt



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

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

# rmmod myext2



1. **修改文件系统操作**

对于mknod函数，我们在myext2中作如下修改：

fs/myext2/namei.c

static int myext2\_mknod (struct inode \* dir, struct dentry \*dentry, int mode, int rdev)

{

printk(KERN\_ERR “haha, mknod is not supported by myext2! you’ve been cheated!\n”);

return -EPERM;

/\*

…..

把其它代码注释

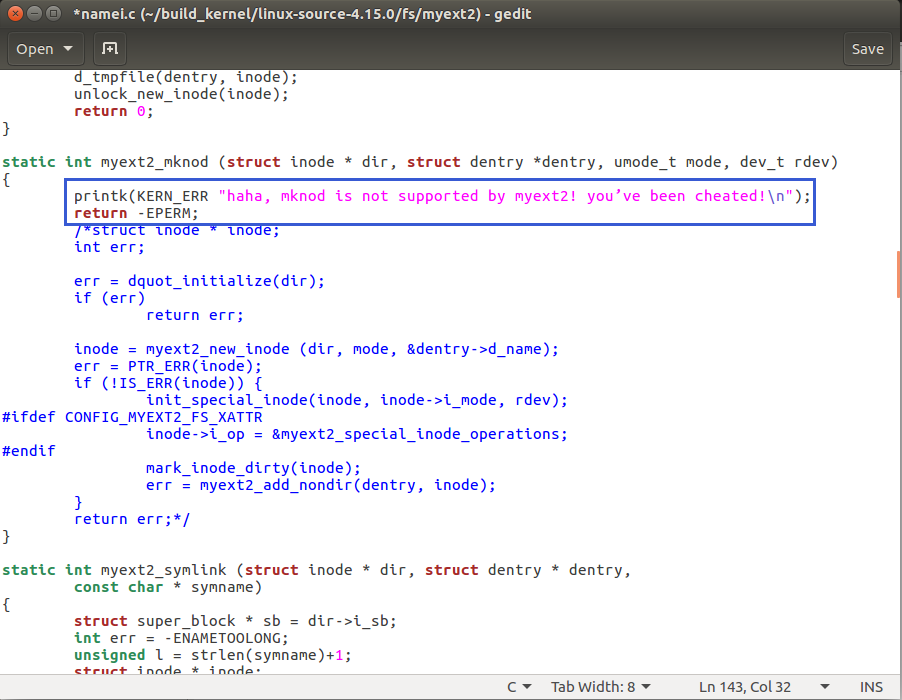
\*/

}

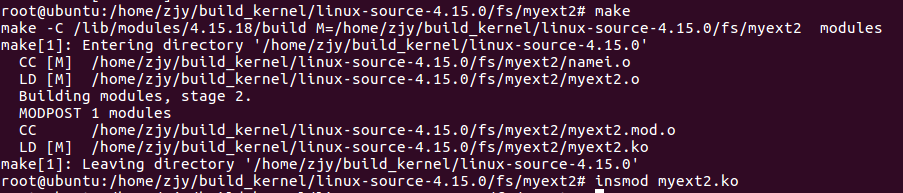
添加的程序中：

第一行 打印信息，说明mknod操作不被支持。

第二行 将错误号为EPERM的结果返回给shell，即告诉shell，在myext2文件系统中，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：



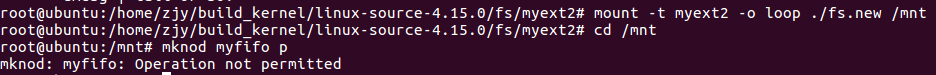
#mount –t myext2 –o loop ./fs.new /mnt

#cd /mnt

#mknod myfifo p

mknod: `myfifo': Operation not permitted

#

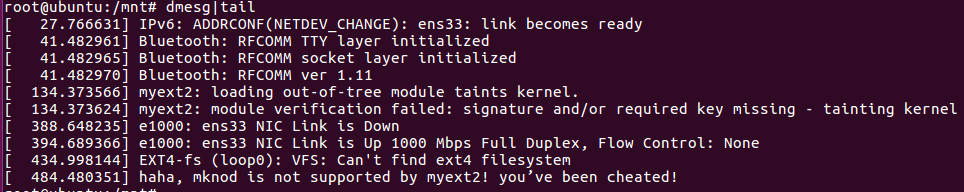


第一行命令：将fs.new mount到/mnt目录下。

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。



1. **添加文件系统创建工具**

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

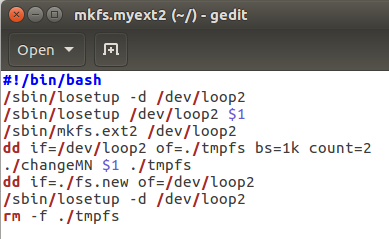
dd if=/dev/loop2 of=./tmpfs bs=1k count=2

./changeMN $1 ./tmpfs

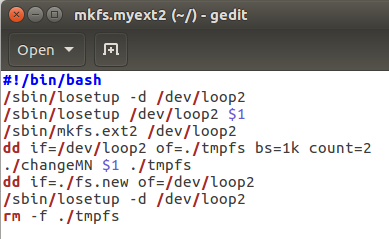
dd if=./fs.new of=/dev/loop2

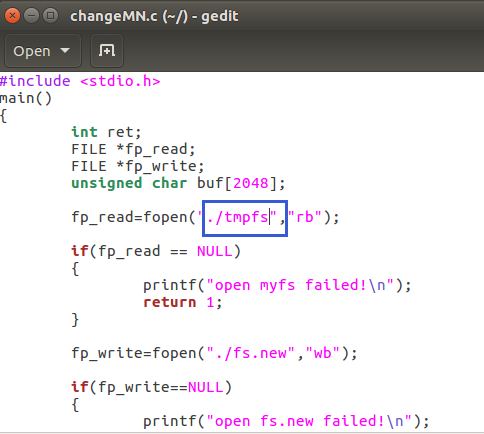
/sbin/losetup -d /dev/loop2

rm -f ./tmpfs



我们发现 mkfs.myext2 脚本中的 changeMN 程序功能，与 4.2 节的 changeMN 功能不一样，下面修改 changeMN.c 程序，以适合本节 mkfs.myext2 和下面测试的需要。





编辑完了之后，做如下测试。

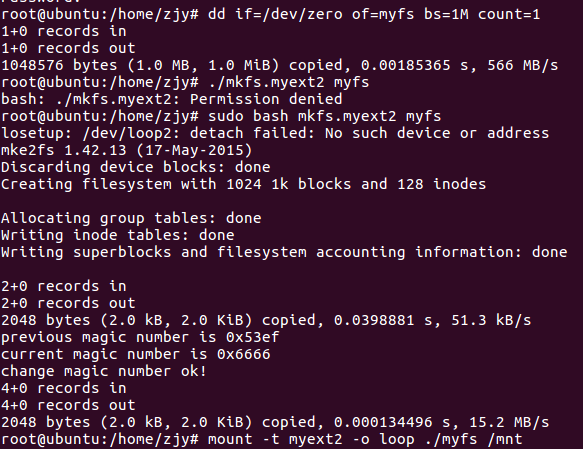
# dd if=/dev/zero of=myfs bs=1M count=1

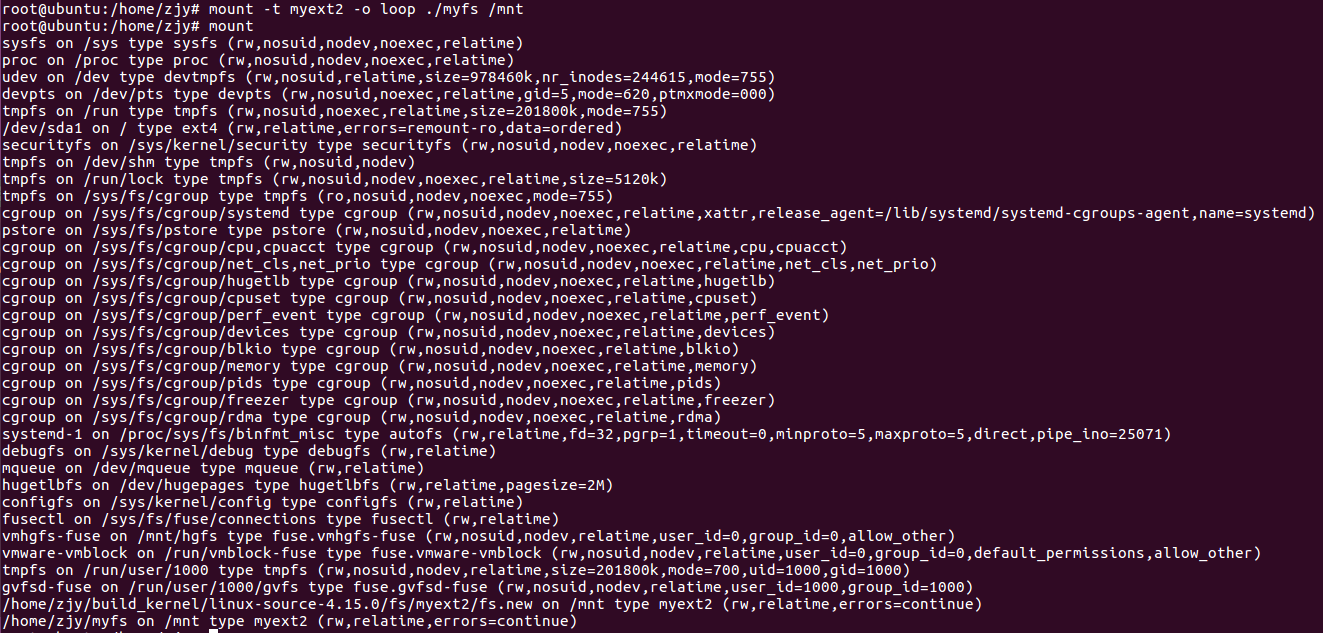
# ./mkfs.myext2 myfs (或 sudo bash mkfs.myext2 myfs )

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

# mount

/dev/loop on /mnt myext2 (rw)

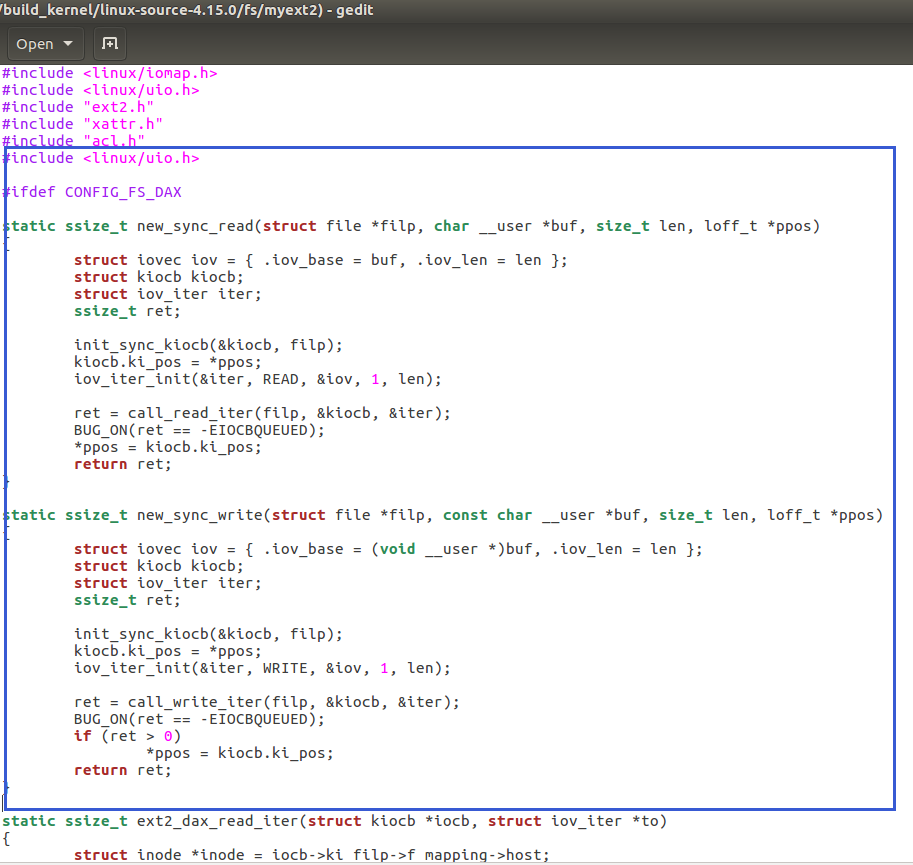


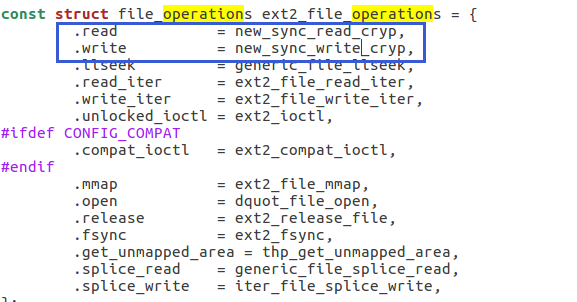


1. **修改加密文件系统的read和write操作**

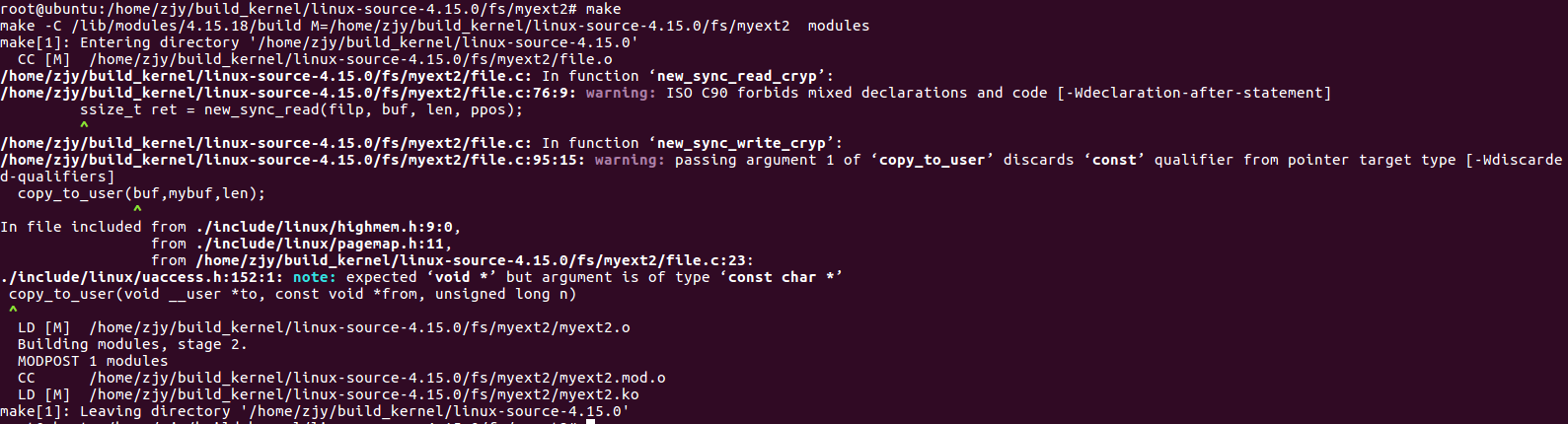
在内核模块 myext2.ko 中修改 file.c 的代码，添加两个函数 new\_sync\_read\_crypt 和 new\_sync\_read\_crypt ， 将 这 两 个 函 数 指 针 赋 给 myext2\_file\_operations 结构中的 read 和 write 操作。在 new\_sync\_write\_crypt 中 增加对用户传入数据 buf 的加密，在 new\_sync\_read\_crypt 中增加解密。可以使用 DES 等加密和解密算法。

首先把 fs/read\_write.c 中的 new\_sync\_read 和 new\_sync\_write 两个函数复制到 file.c 中，再添加两个新函数。



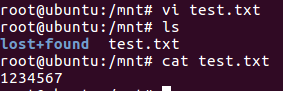


上述修改完成后，再用make重新编译myext2模块，使用命令insmod安装编译好的myext2.ko内核模块。重新加载myext2内核模块，创建一个myext2文件系统，并尝试往文件系统中写入一个字符串文件。





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 。



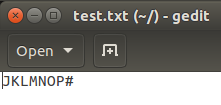
把test.txt文件复制到主目录下：cp test.txt ~ 。

在主目录下打开test.txt文件，查看test.txt文件内容的结果：

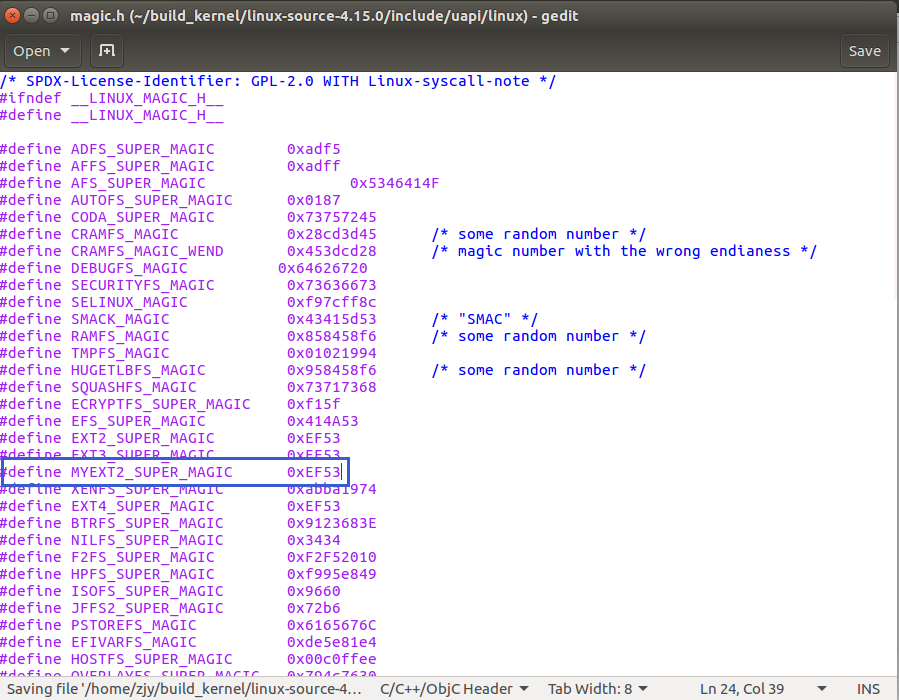




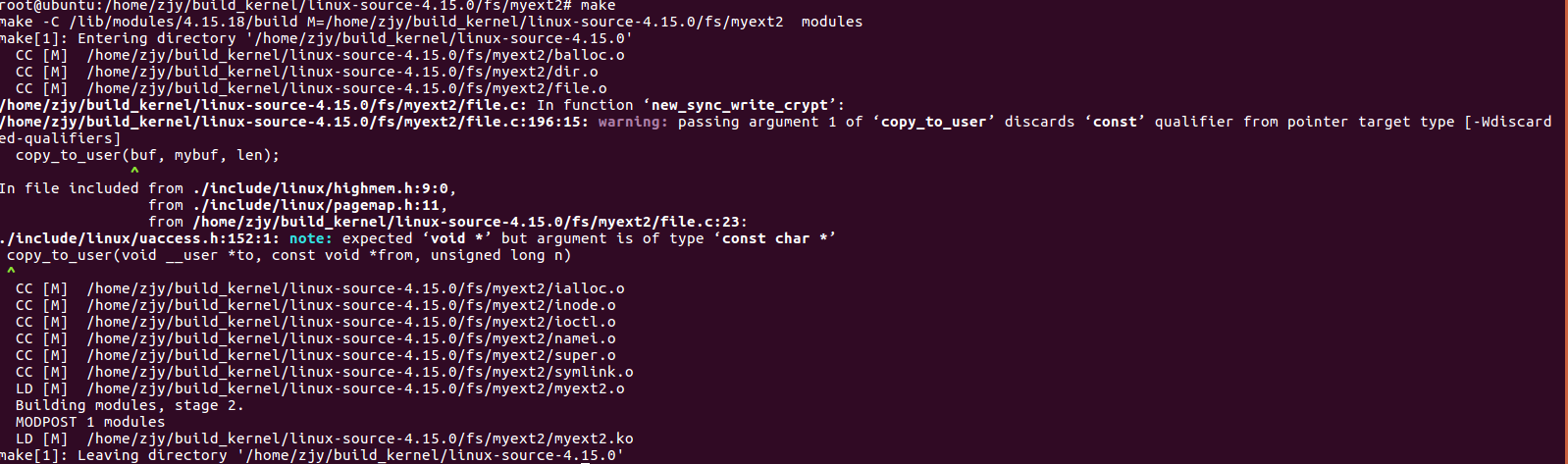
使用文件管理器的复制，再查看结果：



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

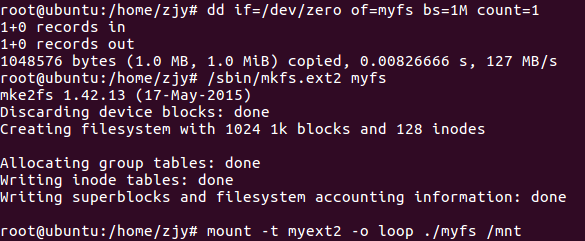
cd

umount /mnt

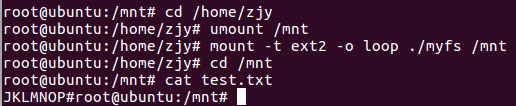
mount -t ext2 -o loop ./myfs /mnt

cd /mnt

cat test.txt







结果分析：

Shell的cp指令复制文件是不加密的。而利用文件系统管理器拷贝会将文件加密的。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

源程序：

changeMN.c

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

new\_sync\_read\_crypt

|  |
| --- |
| static ssize\_t new\_sync\_read\_crypt(struct file \*filp, char \_\_user \*buf, size \_t len, loff\_t \*ppos)  {  int i;  char\* mybuf = (char\*)kmalloc(sizeof(char)\*len,GFP\_KERNEL);  ssize\_t ret = new\_sync\_read(filp, buf, len, ppos);  copy\_from\_user(mybuf, buf, len);  for(i = 0;i < len;i++)  {  mybuf[i] = (mybuf[i] - 25 + 128) % 128;  }  copy\_to\_user(buf, mybuf, len);  printk("haha encrypt %ld\n", len);  return ret;  } |

new\_sync\_write\_crypt

|  |
| --- |
| static ssize\_t new\_sync\_write\_crypt(struct file \*filp, const char \_\_user \*bu f, size\_t len, loff\_t \*ppos)  {  char\* mybuf = (char\*)kmalloc(sizeof(char)\*len,GFP\_KERNEL);  int i;  copy\_from\_user(mybuf, buf, len);  for(i = 0;i < len;i++)  {  mybuf[i] = (mybuf[i] + 25) % 128;  }  copy\_to\_user(buf, mybuf, len);  printk("haha encrypt %ld\n", len);  return new\_sync\_write(filp, buf, len, ppos);  } |

1. 讨论、心得（20分）
2. 这个实验做了三遍，结果发现第一遍做的就是正确的，但是后面几遍实验让我加深了对实验的理解。之前没有弄明白每个指令应该在哪一个父目录下运行，就有些步骤做错了。
3. 我感觉直接从外面复制到命令行的指令会有字符的问题。一模一样的指令，有的可以运行，有的就会报格式错误。
4. 我在myext2执行脚本之后，文件夹内的文件就全变成只读文件了，又因为后面的实验要进行修改操作，在命令行中用vi编辑比较麻烦，因此chmod 666改了一下权限。
5. Shell中的cp将首先读取文件中的数据，然后写到新位置的文件中去，因此当我们从文件系统复制文件时，myext2将被挂载在/mnt，而其他的文件系统在ubuntu上正常运行。我们先解密数据，然后写回，磁盘中的数据存储就不会加密。而利用文件系统管理器拷贝的副本使用mmap而不是读取，直接映射数据，因此是加密的。